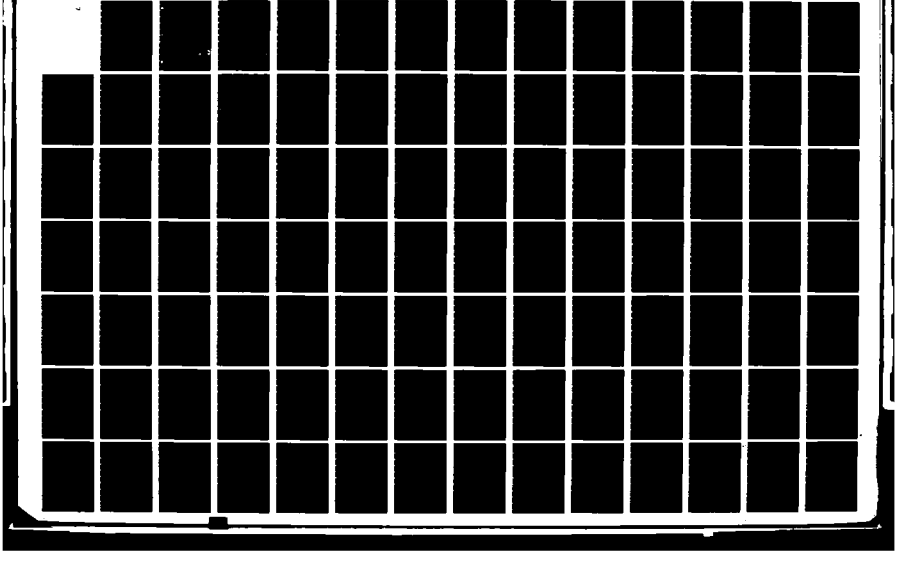
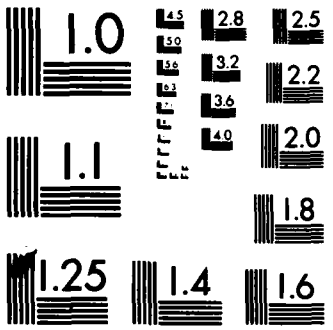
AD-A138 117 THE OPERATOR MAPPING BETWEEN RELATIONAL ALGEBRA 1/2
OPERATORS AND CODASYL BAS..(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. D J NICELY
UNCLASSIFIED DEC 83 AFIT/GCS/EE/83D-15 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

THE OPERATOR MAPPING BETWEEN RELATIONAL ALGEBRA OPERATORS AND CODASYL BASED DATABASES MANAGED BY A CODASYL DBMS

THESIS

AFIT/GCS/EE/83D-15 Debra J. Nicely
Capt USAF

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY (ATC)
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

THE OPERATOR MAPPING BETWEEN RELATIONAL ALGEBRA OPERATORS AND CODASYL BASED DATABASES MANAGED BY A CODASYL DBMS

THESIS

AFIT/GCS/EE/83D-15 Debra J. Nicely
Capt USAF

Approved for public release; distribution unlimited.

AFIT/GCS/EE/83D-15

THE OPERATOR MAPPING BETWEEN RELATIONAL ALGEBRA OPERATORS AND CODASYL BASED DATABASES MANAGED BY A CODASYL DBMS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

Debra J. Nicely, B.S.

Capt USAF

Graduate Computer Science

December 1983

Approved for public release; distribution unlimited.

Preface

This investigation demonstrates one method of mapping relational algebra operators to a network database. It does this by defining relations on a network model in terms of network operators.

I wish to thank James S. Pickney and Captain Brock D. Morris of the ASD Computer Center for their many hours of patience and their assistance with IBM Job Control Language and using IDMS.

I wish to express my sincere gratitude to my thesis advisor, Major Charles Lillie, for his direction and encouragement throughout the entire thesis effort. I also thank Dr. Henry Potoczny for reviewing my thesis and aiding in my understanding of the relational database model.

Finally, I would like to thank my husband (and fellow AFIT student) Captain Robert E. Wilson for his many hours of listening and helpful suggestions on this effort and a special thank you for his support and understanding.

Contents

		Page
Preface		ii
List of Figures		iv
Abstract		vii
I.	Introduction	I-1
	Background	-1
	Relational Database Model	-3
	Relational Data Structure	-3
	Relational Operators	-4
	Network Database Model	-5
	Network Data Structure	-5
	Network Operators	-6
	Comparing Relational and Network	-7
	Data Structures	-7
	Operators	-8
	Problem Statement	-8
	Scope	-9
	Assumptions	-9
	Summary of Current Knowledge	-10
	Hardware Solutions	-10
	Optimizing	-12
	Functional Dependencies	-13
	Approach	-14
II	Requirements	II-1
	Overall Requirements	-1
	Relation Definititions	-1
	Tuple Definitions	-3
	Data Structure Mapping	-3
	Operator Mapping	-13
	Select Mapping	-14
	Project Mapping	-14
	Join Mapping	-15
	Dealing with Duplicates	-15
	Data Dictionary Requirements	-20
	Input/Output Requirements	-20
III	System Design	III-1
	General Design Criteria	-1
	Generic Mapping Program	-2
	Batch Input	-2
	Data Structure Mapping	-8
	Network Structure	-8
	Data Dictionary	-9

		Page
	Mapping Operators	-10
	Generic Data Retrieval Program	-12
	Batch Input	-12
	Data Structure Mapping	-13
	Query Implementation	-14
	Handling Duplicates	-15
	Output	-15
IV	Implementation	IV-1
	General Implementation	-1
	Generic Mapping Program	-2
	Data Structure Mapping	-2
	Network Structure	-3
	Data Dictionary	-3
	Mapping Operators	-4
	Generic Data Retrieval Program	-4
	Query Retrieval	-5
	Data Structure Mapping	-5
	Query Implementation	-5
	Select Implementation	-6
	Project Implemetation	-7
	Join Implemetation	-7
	Handling Duplicates	-8
	Output	-8
V	Testing	V-1
	Testing Database Description	-1
	Generic Mapping Program	-2
	Tested Data Structures	-4
	Structures Not Tested	-4
	Generic Data Retrieval Program	-4
	Syntax Checks	-5
VI	Conclusions and Recommendations	VI-1
	Conclusions	-1
	Network Improvements	-1
	Relational Improvements	-2
	Recommendations	-2
	Mapping Program Enhancements	-3
	Data Retrieval Program Enhancements	-3
Bibliography		BIB-1
Appendix A:	Relational Algebra Syntax	A-1
Appendix B:	Schema	B-1
Appendix C:	Subschema	C-1

		Page
Appendix D:	Data Structure Mappping Operators . . .	D-1
Appendix E:	Data Structure Charts	E-1
Appendix F:	Data Dictionary	F-1
Appendix G:	Batch Input Syntax	G-1
Appendix H:	Table Limitations	H-1
Appendix I:	Test Plan	I-1

List of Figures

Figure		Page
II-1	Valid and Invalid Network Relation Structure	II-2
II-2	CODASYL Record Descriptions	II-4
II-3	CODASYL Record Occurrence Linkage	II-5
II-4	CODASYL Operators to Navigate Database . . .	II-7
II-5	Relation ABC	II-8
II-6	Mapppings for Different Record Types	II-9
II-7	CODASYL Operators to Navigate Database Starting at B	II-10
II-8	CODASYL Subschema Mapping to CODASYL Operators	II-11
II-9	Example Database Bachman Diagrams	II-12
II-10	Input Relation With Same Owner	II-16
II-11	Result Relation With Same Owner	II-17
II-12	Input Relation With Same Member	II-18
II-13	Result Relation With Same Member	II-19
III-1	Data Structure Charts	III-3
III-2	Data Dictionary Bachman Diagrams	III-4
III-3	Bachman Diagram of Relation ABC	III-6
III-4	Relation ABC Batch Input Cards	III-7
III-5	IDMS Operators Needed	III-11
III-6	Data Dictionary Table	III-12
III-7	Example Query Batch Input Cards	III-13
V-1	Test Database Bachman Diagrams	V-2
V-2	Test Relation Structure	V-3
VI-1	Result of a Cartesian Product	VI-5

AFIT/GCS/EE/83D-15

Abstract

This thesis is an example of a mapping of a relational algebra query onto a network database. It consists of the requirements, definition, design and implementation of two generic COBOL programs for implementing such a mapping.

The first program STOREMAP uses a batch input file to build a data dictionary, on the original network database, which defines the relations on which relational algebra queries may be made. This input file is created by the Data Base Administrator who is the most knowledgeable of the structure of the network database and the relations which would be useful to the database's users.

The second program NETTOREL uses the defined relations in the data dictionary and relational algebra queries created by a user to generate a result relations. Data to be included in a result relation is determined by the data dictionary's definition of the relations contained in an associated query and the criteria set by that query.

This original effort shows that the theory for such an operator mapping is valid. Further efforts would be needed to make this implementation user friendly and therefore useful.

CHAPTER I

INTRODUCTION

1.1 BACKGROUND

The objective of a database management system is to centralize control of operational data for computer users. This centralized control reduces the amount of redundant information stored, avoids inconsistency, allows data to be shared, security measures to be applied and integrity to be maintained [Date 1981]

Data independence is an additional benefit desired from databases. It gives applications immunity from changes in the way data is stored and accessed. Databases provide data independence by providing an interface between the application software and the operating system.

Different levels of abstraction from the physical storage and access of data provide different levels of data independence. The higher the level of abstraction the less the user has to know about the structure of the data and the smaller the set of operators required to perform data access

and storage. A higher level of abstraction is more "user friendly " allowing the user to learn less for data processing.

Conversely, the lower the level of abstraction the more control the user has over the data. Applications taking advantage of that control are more efficient, but this requires that the user have a strong grasp on the structure of the data. The lower the level of abstraction the closer it is to the data access which means the larger the set of options which can be tailored to a user's specific need. This is usually done by an application program and requires an application programmer to generate an interface for the user.

There are currently three database models available: (1) hierarchical, (2) network and (3) relational. Hierarchical and network are on the same level of abstraction and require an interface in the form of an application program. Relational is at a high level and allows the user to formulate his own relational queries. This thesis research is exploring the possibility of using a network application program to act as an interface between the data and user, thereby providing the user with relational view of a network database which is more "user friendly".

1.1.1 RELATIONAL DATABASE MODEL

The objective of the relational database model as defined by Codd, is "to provide a sharp and clear boundary between the logical and physical aspects of database management, ... to make the model structurally simple ... and to introduce a high level of language concepts to enable users to express operations upon large chunks of information at a time" [Codd 1982].

The relational model uses the concept of associative addressing which allows the value of a data item to identify its inclusion in results. Associative memory allows multiple vertical data cells to be tested for a particular value simultaneously and each set of horizontal cells which pass the test is marked. Then the marked cells are moved for further processing. The contents of the cells rather then their location determines their possible inclusion in a result area.

The relational database model uses this same concept of data value determining inclusion as opposed to data location. For example what row a tuple is in a relation has no effect on a select operation, rather it is the value of a particular data item for a particular attribute which causes a tuple to be included. Associative addressing makes the database independent of the physical location of the data.

1.1.1.1 RELATIONAL DATA STRUCTURE

The data structure of the relational model is a two dimensional table called a relation. It consists of tuples which make up the rows in the table and attributes which make up the columns. The inter-relationship of the data is established by its value. That value can be used to represent relationships with data from other domains dependent on their groupings. There is also an implied relationship among data items when they are defined as being contained in the same tuple or attribute.

For example, two tuples residing in two different relations which have the same value for a given data item can be used to establish a relationship among all the data items within both tuples. The value implies the relationship as opposed to an explicit definition. There are explicit relationships defined by functional dependencies, which can be used to define a relational database, but these are not permanent parts of the database. Rather, they are used to define the logical structure of the database and become implied relationships once the design is implemented.

1.1.1.2 RELATIONAL OPERATORS

The relational model has a very small number of operators because it operates on large chunks of information. This is part of what makes the relational

model so "user friendly". The relational model provides a wide choice of underlying data access structures since it is at so high a level of abstraction. A poor choice for the underlying structure can result in very slow access time and wasted memory.

Current relational model implementations do not try to bridge the wide gap between the relational data structure and data access. They do not provide a way of storing explicit information about the relationships of specific data which could be used for logical addressing in a direct access world. This is why relational model implementations are often viewed as inefficient. When technology can provide inexpensive associative addressing hardware such a bridge will not be needed but due to current cost factors that will not be in the near future. If relational databases are to be competitive a bridge between relations and direct access must be found.

1.1.2 NETWORK DATABASE MODEL

The Conference on Data Systems Language (CODASYL) is the standard used for the network model. CODASYL was designed by committee (CODASYL Data Base Task Group). It is based on a two-level set or tree structure, with the programmer navigating through the database. This requires a knowledge of the data structure and an understanding of how to navigate it. It also provides the programmer with a high

degree of control which allows for more efficiency. A network database can only be accessed with an application program.

1.1.2.1 NETWORK DATA STRUCTURE

The network data structure is a series of record types which maybe linked together via a series of set types. These links may be one-to-many or many-to-many. A network is very versatile and also can become very complicated. It's level of abstraction is closer to the operating system then to the user and requires the interface of an application program to access the data.

There are implied relationships within a network data structure, such as, two data types being stored in the same record type. There are also explicit relationships as represented in the set types which defines which record types own which member record types. It is these explicit relationships which allow for the tight control over the data and give the application programmer the ability to define the types of direct addressing to be used for data access.

1.1.2.2 NETWORK OPERATORS

Network operators deal with records and sets. There is no provision for accessing multiple records with a single operator. In fact a single operator may be used in a

variety of ways (e.g. there are six different "find" statements). A series of these operators can be used to form a path through the database. These paths are defined by the application programs.

The efficiency of the database is dependent on the quality of the design. Currently there are no straightforward rules, such as functional dependencies, for measuring the quality of the design. Also the database may be tailored to be very efficient for one application at the expense of other applications. This is a result of design choices regarding pointer establishment.

1.1.3 COMPARING RELATIONAL AND NETWORK

The relational model is at a higher level of abstraction than the network model and, as such, is closer to the user. The relational model has complete data independence by using associative addressing. This higher level of abstraction makes the relational model more user friendly than the network model. The network model is on a level of abstraction closer to the data storage medium. The network model uses physical addressing which allows it more control over data storage and access and it is more efficient then the relational model.

1.1.3.1 DATA STRUCTURES

The relational model deals with a two dimensional table data structure. This structure is very familiar to people

and one that is dealt with frequently. This familiarity is part of what makes the relational model easier to learn and understand. The network model uses a collection of one dimensional record types which are linked together via set types which form paths through the data. These paths can become very complicated and are often hard for an application programmer to understand, let alone an unsophisticated user.

1.1.3.2 OPERATORS

Relational operators deal with larger chunks of information than the network operators and have a smaller, easier to learn set of operators. The relational operators are also closer to English sentences which makes them more useful in the area of artificial intelligence. The network operators have more control over the access and storage of data and also have more versatility then the relational model operators. The network model is currently used more then the relational model.

1.2 PROBLEM STATEMENT

A genuine need exists for a readily usable database model that provides immediate access of nonredundant information.

The end users' need for new application programs is exceeding the capacity of data processing departments.

Because the relational model is easy to understand it allows the end user to bypass the application programmer and formulate their own queries to access their data [Codd 1979]. But due to the present need to retrieve data through physical addressing, many relational databases are inefficient.

Network databases are currently selected by commercial users because they are more efficient, more economically prominent and have more stable software then the relational model [Fallahzadeh 1982]. The complexity of the network model requires an application programmer to navigate the database for the user.

1.3 SCOPE

The scope of this thesis is to use the network model to bridge the gap between the relational model and the storage and access of the data. This is done by mapping the relational data structure onto a network database. This mapping is implemented using network operators. Additional network operators can then be embedded in these mappings to perform the relational operations.

An Integrated Database Management System (IDMS) CODASYL model was used for the network model and the mapping to the relational structure is stored within the database. Relational algebra queries are translated into IDMS operators and then embedded into the relational structure

mappings. Results are displayed as a relational table.

1.4 ASSUMPTIONS

The relational algebra operators are those defined by E. F. Codd in his 1970 pioneering paper [Codd 1970]. The IDMS operators will be limited to those defined by the CODASYL Data Base Task Group in their April 1971 report [Data Base Task Group 1971].

The network application program produced in this thesis effort will be truly useful if it can interface with any IDMS database. This requires converting IDMS Data Manipulation Language (DML) statements into COBOL statements without using a precompiler.

1.5 SUMMARY OF CURRENT KNOWLEDGE

Although the inefficiency of relational databases is still being debated, many solutions have been proposed to make it more efficient. Solutions for decreasing access and storage time have included new hardware, taking advantage of the associative addressing of the model, and the optimization of queries. Functional dependencies are used to reduce data redundancy.

1.5.1 HARDWARE SOLUTIONS

Hardware solutions use associative memory processors to allow simultaneous queries on rows within tables. The three basic approaches have been logic-per-track, high-speed

processors, and special-purpose peripherals. Unfortunately under current technological constraints all three are prohibitively expensive.

Content Address Segment Sequential Memory uses the logic-per-track approach. It consists of a linear array of cells composed of a processing element and a memory element that is able to communicate with adjacent cells [Libovski 1978]. Another logic-per-track system is the Rotating Associative Relational Store. It stores a row from a table across a band of tracks instead of a single track. Each band has logic associated with it [Landon 1978].

A Relational Associative Processor is a high speed processor that uses a relatively small number of parallel processing cells for storing tables. Each cell is composed of a processor and a block addressable memory. The processor handles database definitions, insertions, deletions, updates and retrievals. A controller receives instructions from a host computer. The controller decodes these instructions then broadcasts control sequences to initiate cell execution, and passes retrieved items to the host computer [Ozkarahan 1980].

Content Addressable File Store Hardware (CAFSH) is a special-purpose peripheral. It uses a selector to set up key registers with the key value pairs. A row is removed from disk storage. As a row enters each key register, a latch is set if the value in a column in the row fulfills a

test criteria (larger, smaller or the same) against the value in the key register. The results of the key registers then go to latch comparators. If the condition within the selector was met then the output line will be set to one. All rows whose output lines are a one are sent to the host computer [Babb 1978].

1.5.2 OPTIMIZING

Because the relational database model is at such a high level of abstraction it allows queries to be written under current implementations that take a great deal of time to execute. By using a processor to rephrase the query, execution time could be reduced. This technique is called optimization [Ullman 1982].

There are several optimizing strategies. The most common is using algebraic manipulations to reduce the execution time of queries. Certain relational operations, such as a select, can reduce the size of a relation before doing more costly operations, such as joins. By rephrasing the queries so that these reducing operators are done first, the execution time for the entire query can be reduced.

A lower level of optimization deals with how to use information about the organization of the file holding relational information to minimize access time. This type of optimization is very dependent on the implementation of the relational model used. Most commonly, relational

implementations rely heavily on indices to decrease data access time.

There are problems with both of these types of optimizing. The first takes time to optimize and if the query is already optimal, it will take longer to process. The second strategy increases the amount of memory needed to store relations with the overhead of indices. An increase in storage space is needed to reduce access time.

1.5.3 FUNCTIONAL DEPENDENCIES

Functional dependencies define the relationships among the attributes within a relation. They "are actually assertions about the real world" and should be enforced by a Data Base Management System (DBMS) [Ullman 1982].

Certain relations can be broken down into multiple relations and by doing so reducing the amount of redundant data. Functional dependencies can be used to determine where to break a table without losing information.

Tables can be broken in such a way as to retain all the information but some of the functional dependencies can be lost. The database designer must be aware of this and determine if such a loss is acceptable. If not, not breaking a table over a functional dependency can cause redundant information to be retained. A tradeoff must be made when using functional dependencies to break down

tables. Although using functional dependencies to break down tables allows less data to be stored such breaks requires a costly join across all the smaller relations to retrieve the original relation. The typical tradeoff is more memory for faster access.

1.6 APPROACH

The approach taken consists of defining a way to map from a network data structure to a relational data structure and using that to implement relational algebra queries. A description of how to create such a map presented along with how to modify that map to implement queries. A generic IDMS application program written in COBOL was produced to store this mapping data on any IDMS database to be queried against.

A second generic program was produced to retrieve the data structure mappings and modify them to implement the relational algebra queries. This program has been written in COBOL and does not use the IDMS precompiler for the IDMS DML statements.

For an IDMS record type to be included in a relation it must be connected to at least one other record type in the relation via a set type or must be the only record type in the relation. Each record type must have a mapping defined by the Data Base Administrator (DBA) starting with it and reaching all other record types in the relation.

CHAPTER II

REQUIREMENTS

2.1 OVERALL REQUIREMENTS

This thesis research shows that it is possible to give network database users a relational view of their data. This is a two step process. The first step is to define a mapping between a network data structure and a relational data structure. The second is to use these defined mappings to retrieve the network data and implement a relational query using network operators and display the results in a relational form.

2.2 RELATION DEFINITIONS

All the record types to be included in a relation must be connected by set types. This way pointers are established linking record occurrences together and an explicit relationship among record occurrences is defined (Figure II-1).

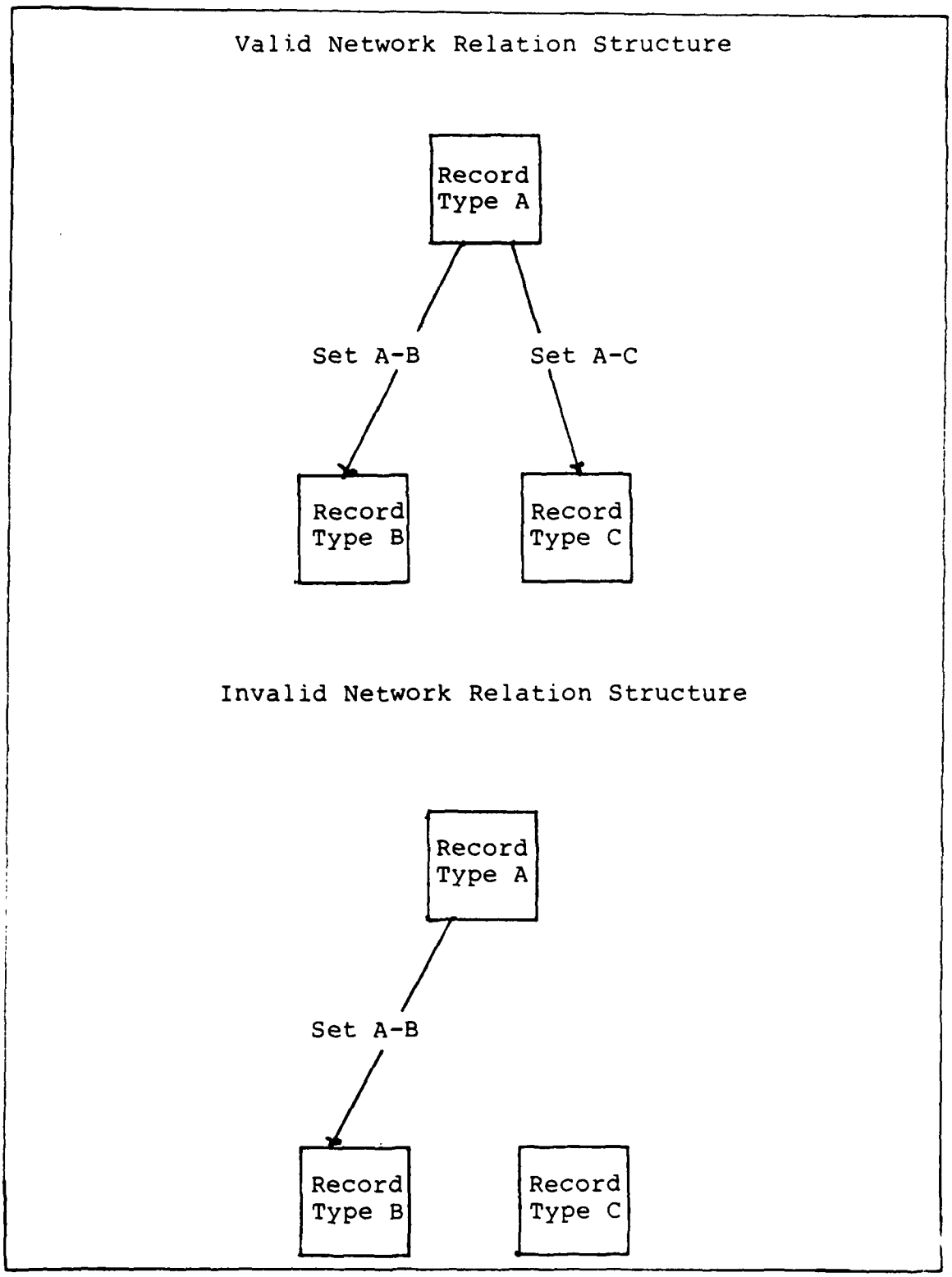


Figure II-1. Valid and Invalid Network Structures

2.2.1 TUPLE DEFINITION

In most cases the data structure path will include a nesting structure. At each level of the path it is possible for a given owner record occurrence to be memberless or a member occurrence to be ownerless. To avoid incomplete data from being included within the result relation, status checks for such occurrences are done after IDMS data retrievals.

2.3 DATA STRUCTURE MAPPING

A relation can be defined in terms of a set of CODASYL record types, set types and operators to define a navigation path through a network database. The sets define the explicit relationships between record types and implicit relations are given by the field definitions of the records.

For example lets look at the valid relational structure used in Figure II-1 and lable it Relation ABC. Figure II-2 defines the record occurences in the record types A, B, and C and the fields contained in the record occurences for each type. Figure II-3 defines the explicit relations of the record occurrences via the set types A-B and A-C. The implied relations are defined by the field values contained in the same record occurrence.

Now using the record occurrence definitions in Figure II-2 and the data relationships defined in Figure II-3 the operators needed to navigate the CODASYL database are given

CODASYL RECORD DESCRIPTIONS FOR RECORD TYPE A

Record Occurrence	Record Contents
A1	a11, a12, a13
A2	a21, a22, a23
A3	a31, a32, a33
A4	a41, a42, a43

CODASYL RECORD DESCRIPTIONS FOR RECORD TYPE B

Record Occurrence	Record Contents
B1	b11, b12, b13
B2	b21, b22, b23
B3	b31, b32, b33
B4	b41, b42, b43
B5	b51, b52, b53
B6	b61, b62, b63

CODASYL RECORD DESCRIPTIONS FOR RECORD TYPE C

Record Occurrence	Record Contents
C1	c11, c12, c13
C2	c21, c22, c23
C3	c31, c32, c33
C4	c41, c42, c43
C5	c51, c52, c53
C6	c61, c62, c63
C7	c71, c72, c73

Figure II-2. CODASYL Record Descriptions

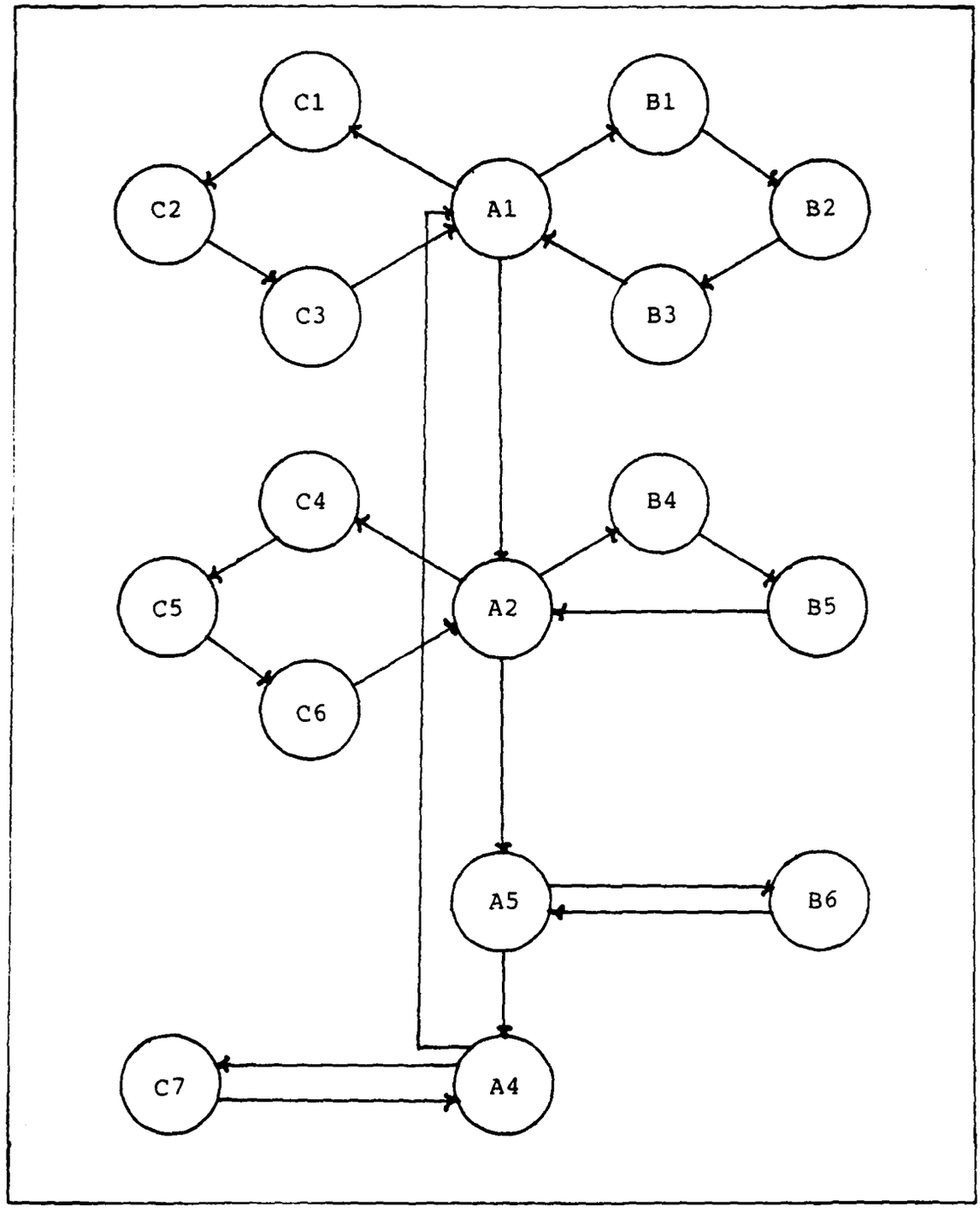


Figure II-3. CODASYL Record Occurrence Linkage

in Figure II-4 and the results of such a navigation is given in Figure II-5. Notice record occurrence A3 does not have an entry in Relation ABC because it has no record type Cs linked to it and likewise record occurrence A4 has no entries since it has no record type Bs linked to it.

A defined navigation path within a CODASYL database maps to a tuple in the relational data structure. Each record type contained in a relation must have a different navigation path defined by the DBA(Figure II-6). These paths are defined in terms of CODASYL operators and are to be stored within the CODASYL database the relation is defined for. A generic data retrieval program uses these paths as a mapping between a network data structure used to access the data and a relational data structure used to display query results.

A different path is needed for each record type in a relation to be used within a given relation to provide for optimal usage of the CODASYL operators for each given query. For example if a select clause is contained within a relational algebra query then the starting point of the path traversal should be the record type which contains the attribute specified in the select operation to avoid a linear search through the CODASYL database. For example Figure II-7 shows the new CODASYL operators needed if the navigation path were to start with Record Type B.

CODASYL Operators	Occurrence Retrieved
OBTAIN FIRST A	A1
OBTAIN FIRST B WITHIN A-B	B1
OBTAIN FIRST C WITHIN A-C	C1
OBTAIN NEXT C WITHIN A-C	C2
OBTAIN NEXT C WITHIN A-C	C3
OBTAIN NEXT B WITHIN A-B	B2
OBTAIN FIRST C WITHIN A-C	C1
OBTAIN NEXT C WITHIN A-C	C2
OBTAIN NEXT C WITHIN A-C	C3
OBTAIN NEXT B WITHIN A-B	B3
OBTAIN FIRST C WITHIN A-C	C1
OBTAIN NEXT C WITHIN A-C	C2
OBTAIN NEXT C WITHIN A-C	C3
OBTAIN NEXT A	A2
OBTAIN FIRST B WITHIN A-B	B4
OBTAIN FIRST C WITHIN A-C	C4
OBTAIN NEXT C WITHIN A-C	C5
OBTAIN NEXT C WITHIN A-C	C6
OBTAIN NEXT B WITHIN A-B	B5
OBTAIN FIRST C WITHIN A-C	C4
OBTAIN NEXT C WITHIN A-C	C5
OBTAIN NEXT C WITHIN A-C	C6
OBTAIN NEXT A	A3
OBTAIN FIRST B WITHIN A-B	B6
OBTAIN FIRST C WITHIN A-C	no record
OBTAIN NEXT A	A4
OBTAIN FIRST B WITHIN A-B	no record

Figure II-4. CODASYL Operators to Navigate Database

a11, a12, a13, b11, b12, b13, c11, c12, c13
a11, a12, a13, b11, b12, b13, c21, c22, c23
a11, a12, a13, b11, b12, b13, c31, c32, c33
a11, a12, a13, b21, b22, b23, c11, c12, c13
a11, a12, a13, b21, b22, b23, c21, c22, c23
a11, a12, a13, b21, b22, b23, c31, c32, c33
a11, a12, a13, b31, b32, b33, c11, c12, c13
a11, a12, a13, b31, b32, b33, c21, c22, c23
a11, a12, a13, b31, b32, b33, c31, c32, c33
a21, a22, a23, b41, b42, b43, c41, c42, c43
a21, a22, a23, b41, b42, b43, c51, c52, c53
a21, a22, a23, b41, b42, b43, c61, c62, c63
a21, a22, a23, b51, b52, b53, c41, c42, c43
a21, a22, a23, b51, b52, b53, c51, c52, c53
a21, a22, a23, b51, b52, b53, c61, c62, c63

Figure II-5. Relation ABC

```
Mapping Beginning With Record Type A

OBTAIN NEXT A
OBTAIN NEXT B WITHIN A-B
OBTAIN NEXT C WITHIN A-C


Mapping Beginning With Record Type B

OBTAIN NEXT B
OBTAIN OWNER WITHIN A-B
OBTAIN NEXT C WITHIN A-C


Mapping Beginning With Record Type C

OBTAIN NEXT C
OBTAIN OWNER WITHIN A-C
OBTAIN NEXT B WITHIN A-B
```

Figure II-6. Mapppings for Different Record Types

CODASYL Operators	Occurrence Retrieved
OBTAIN FIRST B	B1
OBTAIN OWNER WITHIN A-B	A1
OBTAIN FIRST C WITHIN A-C	C1
OBTAIN NEXT C WITHIN A-C	C2
OBTAIN NEXT C WITHIN A-C	C3
OBTAIN NEXT B	B2
OBTAIN OWNER WITHIN A-B	A1
OBTAIN FIRST C WITHIN A-C	C1
OBTAIN NEXT C WITHIN A-C	C2
OBTAIN NEXT C WITHIN A-C	C3
OBTAIN NEXT B	B3
OBTAIN OWNER WITHIN A-B	A1
OBTAIN FIRST C WITHIN A-C	C1
OBTAIN NEXT C WITHIN A-C	C2
OBTAIN NEXT C WITHIN A-C	C3
OBTAIN NEXT B	B4
OBTAIN OWNER WITHIN A-B	A2
OBTAIN FIRST C WITHIN A-C	C4
OBTAIN NEXT C WITHIN A-C	C5
OBTAIN NEXT C WITHIN A-C	C6
OBTAIN NEXT B	B5
OBTAIN OWNER WITHIN A-B	A2
OBTAIN FIRST C WITHIN A-C	C4
OBTAIN NEXT C WITHIN A-C	C5
OBTAIN NEXT C WITHIN A-C	C6
OBTAIN NEXT B	B6
OBTAIN OWNER WITHIN A-B	A3
OBTAIN FIRST C WITHIN A-C	no record
OBTAIN NEXT B	no record

Figure II-7.
CODASYL Operators to Navigate Database Starting at B

This thesis effort explored the possibility of using an application program's subschema to generate the navigation path for a relation. This could have been implemented by a piece of software. For example the subschema SUBSCABC shown in Figure II-8 contains the records and sets used in Relation ABC and the same operators shown in Figure II-6 could have been generated from the subschema and produced the same Relation ABC.

Subschema ABC

```
ADD SUBSCHEMA NAME IS SUBSCABC
OF SCHEMA NAME IS SCHMABC
DMCL NAME IS ABCDMCL.

ADD RECORD A.
ADD RECORD B.
ADD RECORD C.

ADD SET A-B.
ADD SET A-C.
```

Figure II-8.
CODASYL Subschema Mapping to CODASYL Operators

Figure II-9 is an example of a network structure with a loop. The same two record types are in two different set types. In one set type (A.B) record type A owns record type B. In the other set type (B.A) record type B owns record type A. There are three possible relations which could be defined. Those record occurrences in A and B where A owns B (Relation A-B), those record occurrences in A and B where B

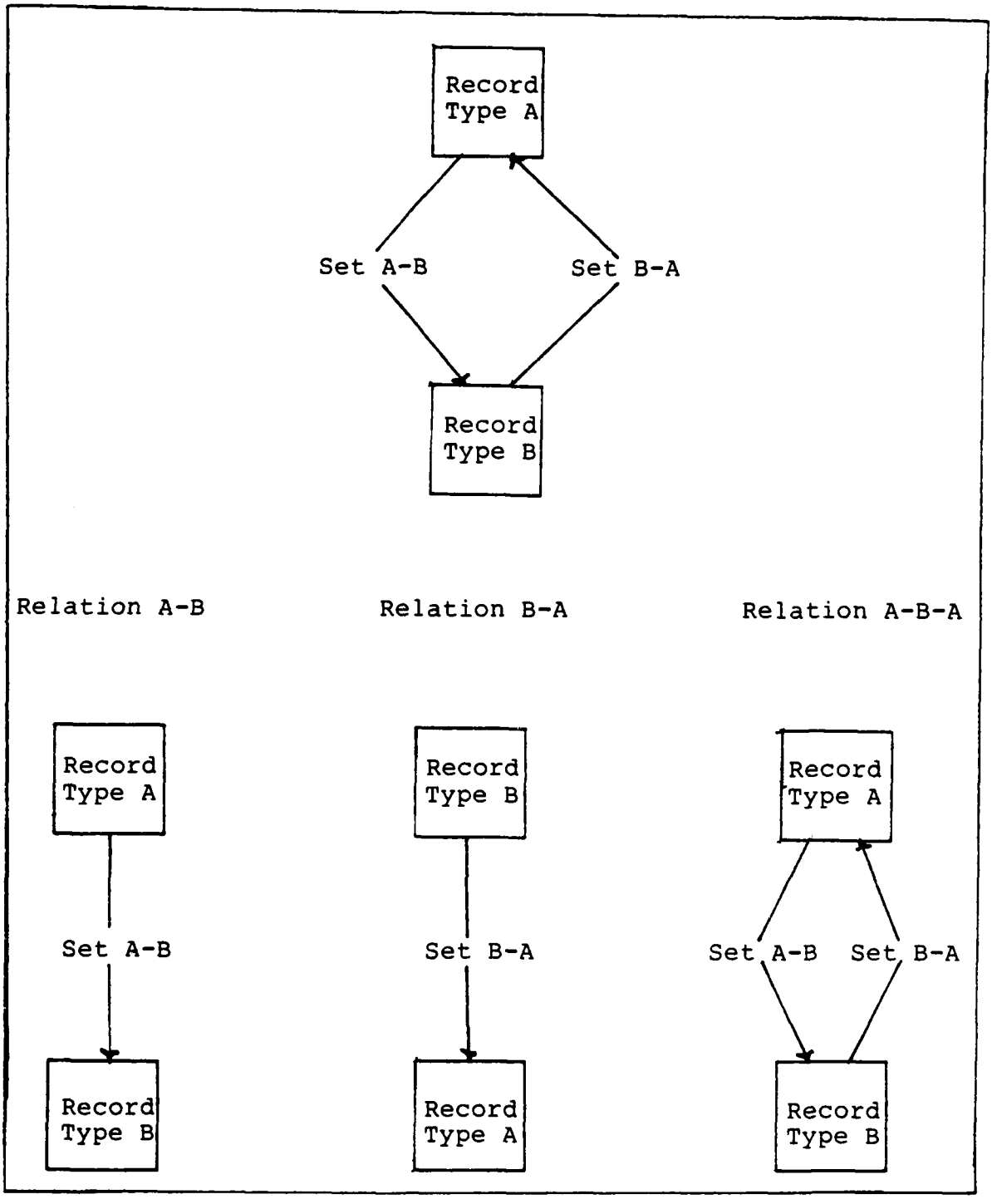


Figure II-9. Example Database Bachman Diagrams

owns A (Relation B-A) or those record occurrence in A and B where A owns B and B owns A (Relation A-B-A).

In the subschema needed for Figure II-9 a decision must be made as to which path or paths should be used requires the intervention of a DBA. This is what precluded using such a subschema to generate the path but it would be useful in subschemas which did not have a looping structure. Time did not permit such an implementation since it would not work for all applications.

The DBA is the one most knowledgeable on the intent of the database. He is the logical choice for defining these mappings. In the above example all three possible relations could be defined, if they would be meaningful, and useful to a user.

In a complicated CODASYL database these data structure mappings will be just as complicated as are currently needed for application programs. But, by defining the paths in terms of relations it will give the user the ability to view the data from different perspectives without requiring a set of tailor made application programs. Also, only the DBA will need to deal with the complexity of the data structure instead of multiple application programmers.

2.4 OPERATOR MAPPINGS

Only the select, project, and join operators are implemented within this research. To implement these

defined result attributes in the result working storage area. The projection operators are implemented by skipping those fields which are not to be projected.

2.4.3 JOIN MAPPING

The join operator has been limited to relations which have a record type in common. If two relations have a record type in common then a set type record in each relation will contain that record since it is a requirement that all record types defined in a relation must be connected via set type. This means the pointers needed to connect the two relations will be present. This allows a single mapping of all the record types in both relations to be created from a mapping already defined from each relation being joined. This means all the record types contained in the mapping as a result of the join will be connected by a set type. Figure II-10 and II-12 are examples of two possible relations that could be joined and Figure II-11 and II-13 are the results of such joins.

2.4.4 DEALING WITH DUPLICATES

According to the definition of a relation duplicate tuples are not allowed. To avoid duplication a sort must be done on the resulting relation and duplicates removed. Duplicates could occur because a record type might be used to retrieve its member records but none of its data

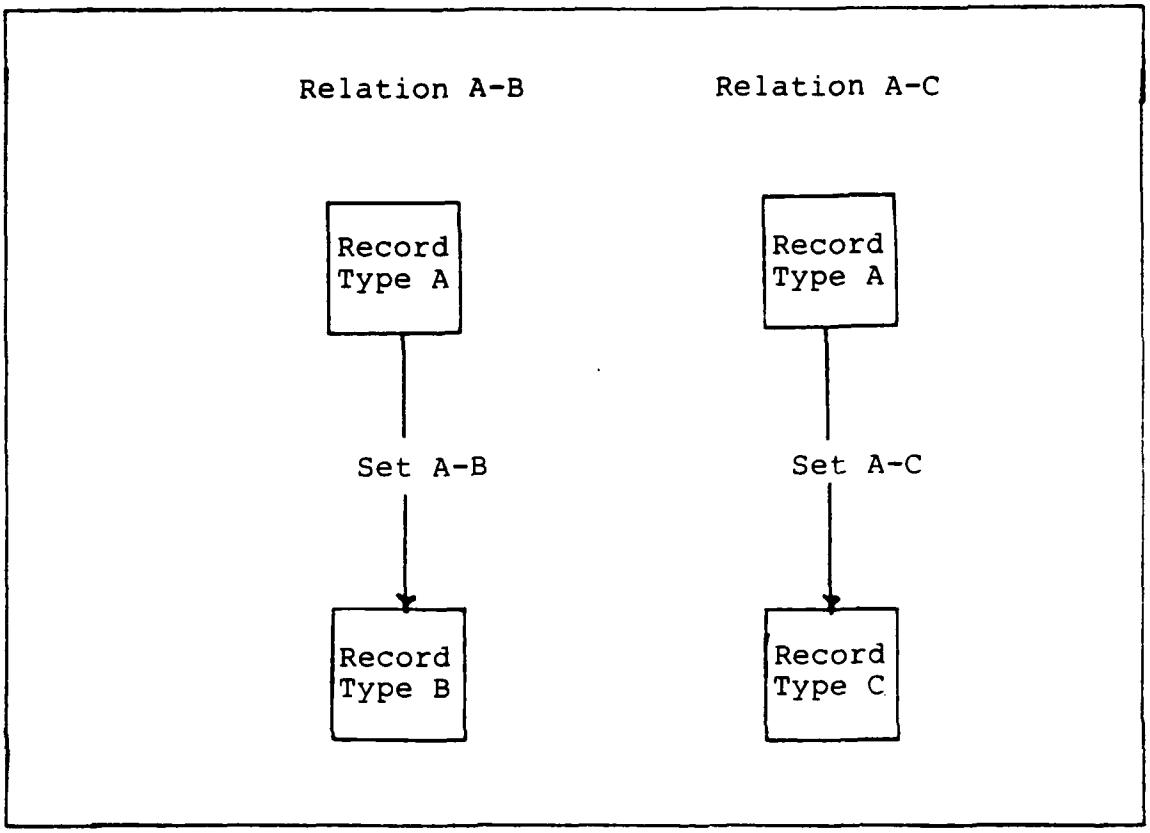


Figure II-10. Input Relations With Same Owner

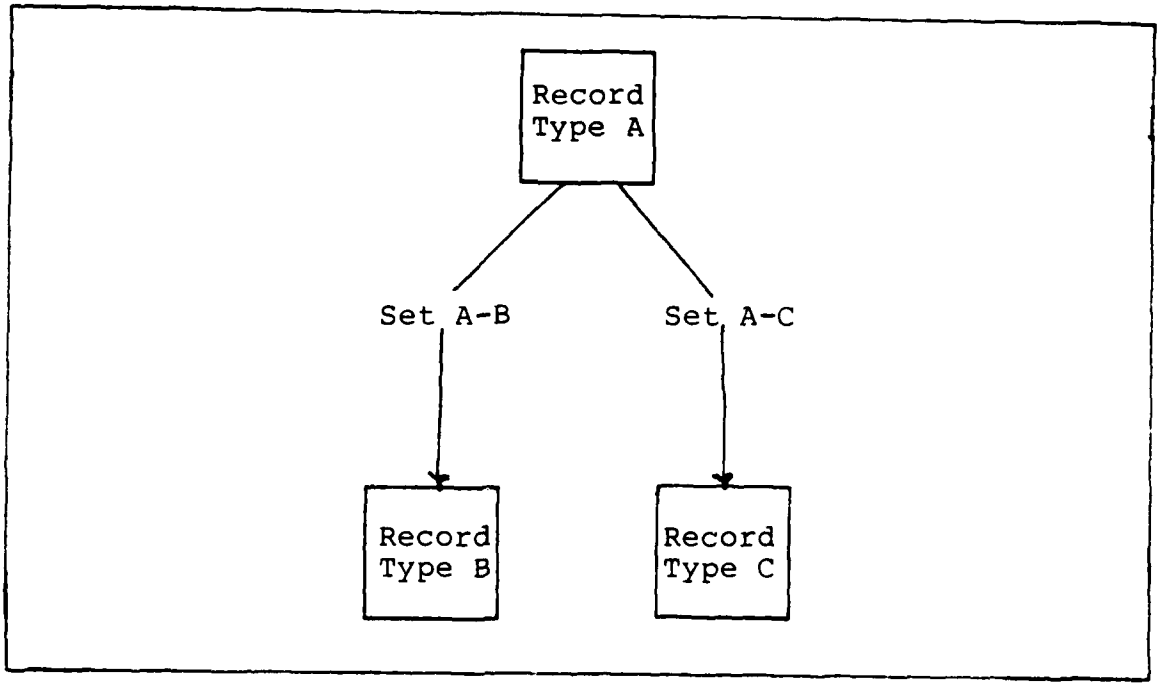


Figure II-11. Result Relation with Same Owner

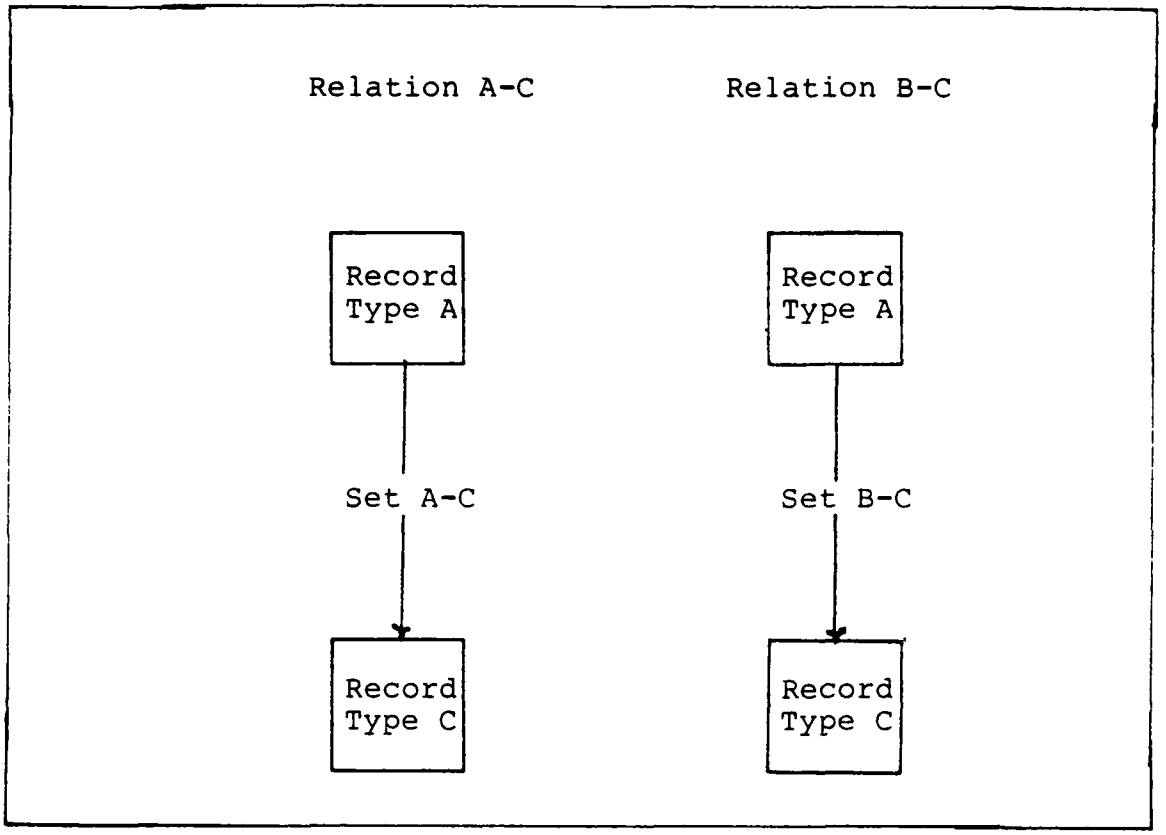


Figure II-12. Input Relations with Same Member

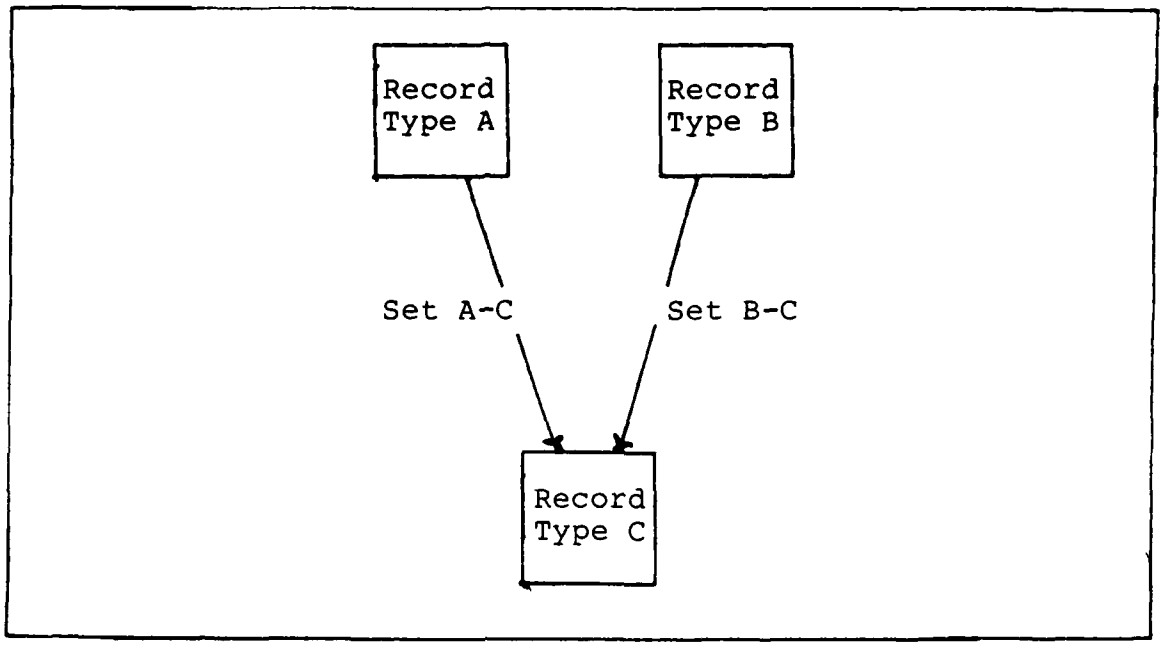


Figure II-13. Result Relation With Same Member

reported allowing two tuples to differ only by the missing data.

2.5 DATA DICTIONARY REQUIREMENTS

The data dictionary for defining relations will be stored in a CODASYL format upon the database being queried. It will contain entries for the record types to be used, the set types, the attribute names and their associated record types and displacement and the navigation path description (data structure mappings) in terms of CODASYL operators.

2.6 INPUT/OUTPUT REQUIREMENTS

All input and output will be done in a batch environment. Relational queries will be according to the format and the symbols defined in Appendix A. The batch input syntax for both generic programs is given in Appendix H. The format of the output file will be determined by the data structure mappings and any project operators included within the relational algebra query. The structure of the output file will be in the form of a relation. Multiple queries on the same database is allowed.

CHAPTER III

SYSTEM DESIGN

3.1 GENERAL DESIGN CRITERIA

Two generic application programs were developed to be used with any IDMS database. One stores the data necessary for defining relations in terms of a network data structure (STOREMAP) and the other implements a relational algebra query (NETTOREL).

STOREMAP and NETTOREL are written as generic programs so they could be used with any IDMS database. They were written in COBOL because the mapping definitions use table indices to identify the sets and areas used and COBOL handles tables very efficiently. COBOL also allows for redefining memory which was used frequently in STOREMAP and NETTOREL.

IDMS was chosen for the network database model because it was the only network model available. A batch input was chosen since it was what was most readily available on the system which had an IDMS capability.

The generic programs STOREMAP and NETTOREL are written in structured code to make them easy to understand and enhance. Figure III-1 contains a high level structure chart of both programs. A complete set of data structure charts for STOREMAP and NETTOREL are contained in Appendix E. A complete data dictionary for both generic programs containing the paragraph and variable names used is maintained on a DBase II relational database. A copy of the relations and the structure used is given in Appendix F.

3.2 GENERIC MAPPING PROGRAM

Before the relational algebra query can be implemented the relations must be defined. Since the underlaying data structure is network the data dictionary and mapping operators are to be stored within each IDMS database to be queried. There is a set network data structures to be used for the dictionary and mapping operators. It is defined in Figure III-2, and the associated subschema modification is given in Appendix C.

3.2.1 BATCH INPUT

All inputs to the mapping structure generic program STOREMAP is in the form of batch input cards. The first nine characters on each card define the card type (SUBSCHEMA, RELATION, RECORD, ATTRIBUTE, SET MAPPING or END).

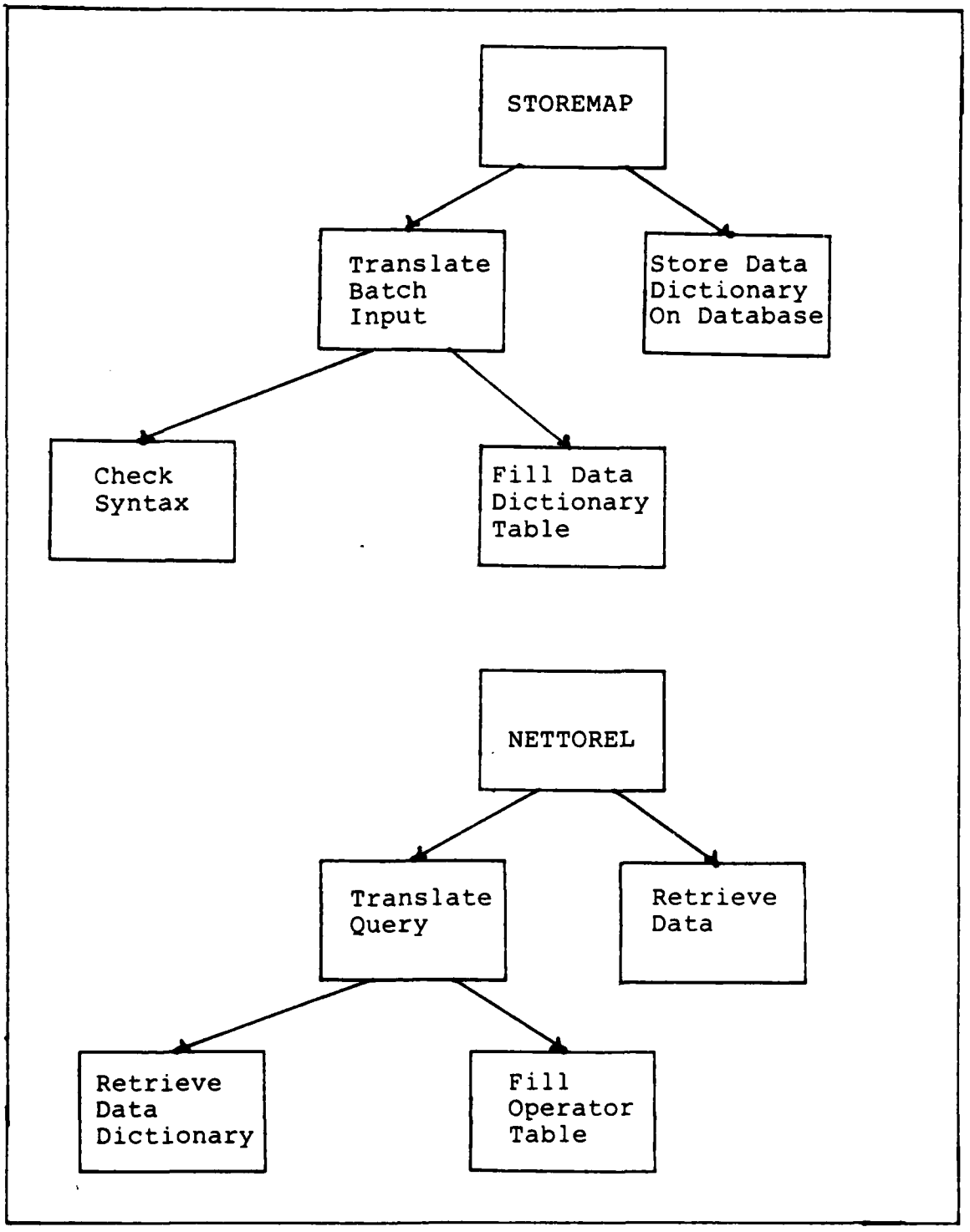


Figure III-1. Data Structure Charts

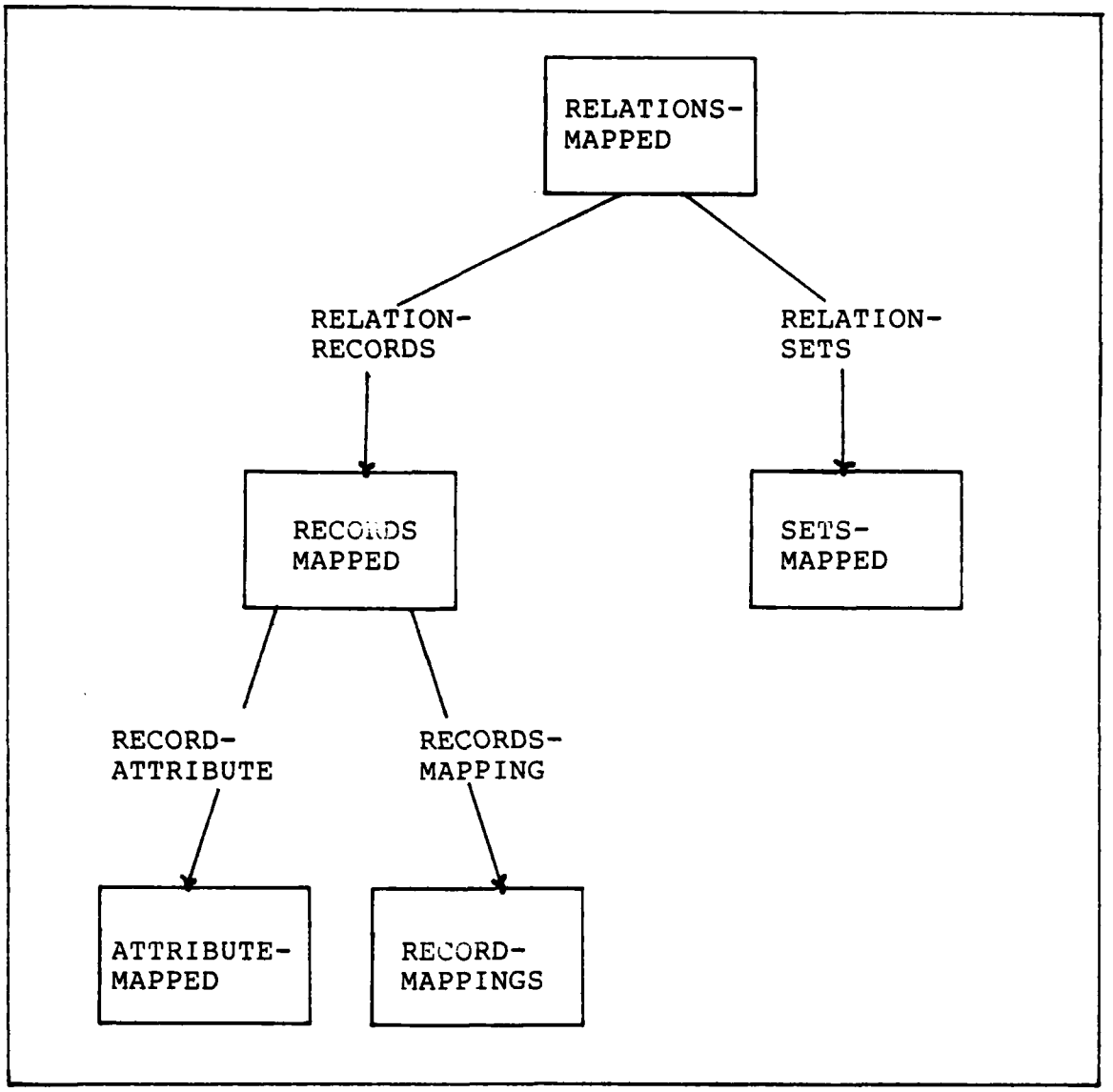


Figure III-2. Data Dictionary Bachman Diagrams

An end card must be placed after a group of set, attribute, and mapping card types. The order of the card types form an implied relation. Figure III-3 is a example relation used for setting up its associated batch cards given in Figure III-4.

As you can see the first card is a subschema card and there can only be one. The next card is a relation card and all cards following it define the relation ABC until the next relation card is encountered or an end of file maker is set.

The next series of cards are the sets contained within the relation ABC. Since there can be more than on set card an end card is needed so STOREMAP will know that the next card will be a record card.

The record card for record type A is followed by a series of attribute cards which defines the attributes contained in relation ABC which are stored on record type A occurrences. Once again an end card is needed to signify that the next card is not an attribute card but rather a mapping card.

The mapping cards that follow are associated only with record type A and the end card signifies that the next card is either a record card, a relation card or an end of file marker. In this example it is a record type card for record B and the definitions for that record type follows.

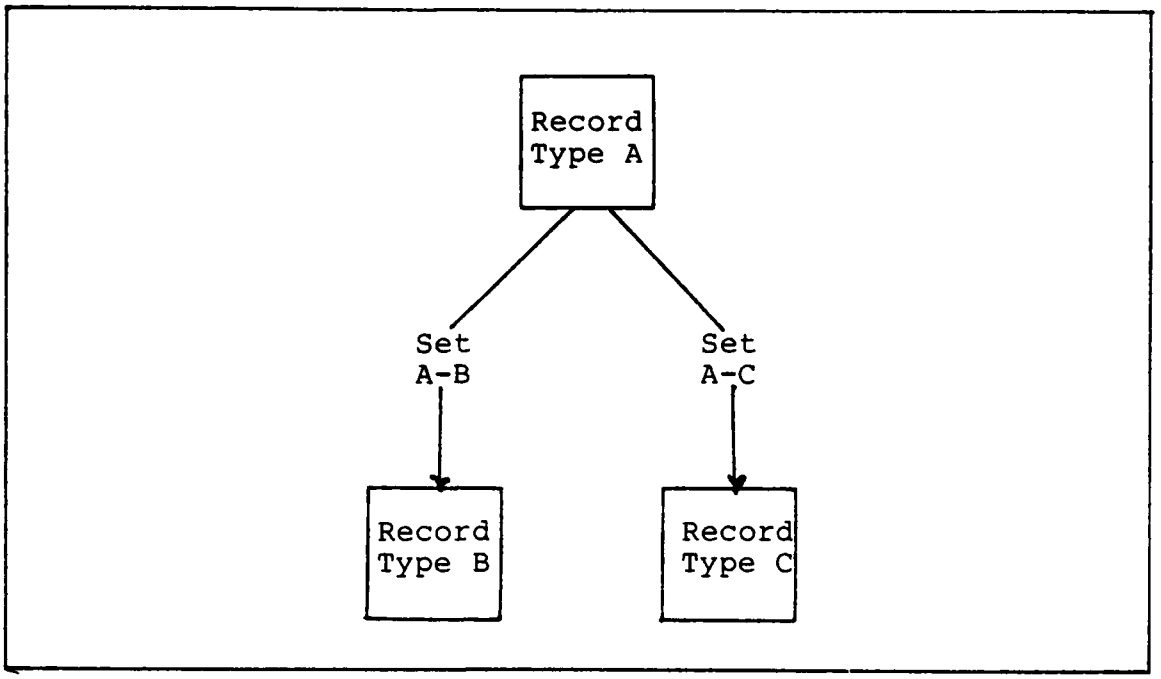


Figure III-3. Bauchman Diagram of Relation ABC

```
SUBSCHEMA ABC-SUB-SCHEMA
RELATION  ABC
SET       A-B
SET       A-C
END
RECORD    A                  REGION-A          0
ATTRIBUTE A1                 00 00 02 X
ATTRIBUTE A2                 02 02 02 X
ATTRIBUTE A3                 04 04 02 X
END
MAPPING   6 A
MAPPING   2 B                  A-B
MAPPING   2 C                  A-C
MAPPING   9
END
RECORD    B                  REGION-A          1
ATTRIBUTE B1                 00 06 02 X
ATTRIBUTE B2                 02 08 02 X
ATTRIBUTE B3                 04 10 02 X
END
MAPPING   6 B
MAPPING   1 A
MAPPING   2 C                  A-C
MAPPING   9
END
RECORD    C                  REGION-A          0
ATTRIBUTE C1                 00 12 02 X
ATTRIBUTE C2                 02 14 02 X
ATTRIBUTE C3                 04 16 02 X
END
MAPPING   6 C
MAPPING   1 A
MAPPING   2 B                  A-B
MAPPING   9
END
```

Figure III-4. Relation ABC Batch Input Cards

3.2.2 DATA STRUCTURE MAPPING

The generic program NETTOREL which implements the relational algebra queries must have a mapping of the network data structure to a relation. This is done with a series of network operators which form a path through the network database. The completion of the path signifies a tuple within the specified relation.

Lets go back to our example relation ABC and assume we have a query to retrieve the relation ABC. Since no selects were requested the first mapping will be used which is the mapping defined for record type A. NETTOREL will retrieve the first occurrences of record type A in the area REGION-A. After a successful retrieval of a record type A record occurrence than the first record type B record occurrence owned by that record type A occurrence is retrieved. Finally the first record type C record occurrence owned by that record type A is retrieved and a tuple is added to the relation. That is assuming that the path was complete and there was at least one record type B owned to the retrieved record type A and at least one record type C was owned to the retrieved record type A.

3.2.3 NETWORK STRUCTURE

The data structure within the network database model is a series of areas which contain record types linked together by set types. These sets define an explicit relationship

between record types and a collection of record types which are all connected by set types are used to form a relation. The sets form a navigation path among the record types. This path may or may not be circular. Either way the relational algebra query determines which record type the path traversal starts with.

Remember in the example there was no select so the navigation started with record type A. If there had been a select such as select B1 where B1 = "b1" then the navigation would have started at record type B then A then C.

In addition since record type B has the calc flag set then instead of using an obtain first record in area operator being used the immediate value "b1" can be moved to the B1 field in the network working storage area and an obtain calc operator used so only record type B occurrences which have a value of "b1" in the B1 field will be retrieved. This is the reason why each record type must have its own navigation path defined and stored on the data dictionary.

3.2.4 DATA DICTIONARY

The records in the network to relational data structure mappings form the data dictionary for the relational model. The RELATIONS-MAPPED record contains the name of the relation to be used in relational algebra queries each record occurrence of RELATIONS-MAPPED has a pointer to the

sets and a pointer to the records that it owns (Figure III-2).

Each occurrence of RECORDS-MAPPED contains the record type name used by the network model, the area it is stored in and a flag if this record type can be reached via a calc operator. It also has a pointer to the ATTRIBUTE-MAPPED record occurrences it owns and a pointer to the RECORD-MAPPINGS it owns.

ATTRIBUTE-MAPPED contains the relational attribute names associated with the network fields used in that record type and their field displacement, length and data type. Its pointers are to the other attributes owned by the same RECORDS-MAPPED.

RECORD-MAPPINGS contains the op-code needed to translate the network operator for the data retrieval, the record type to be retrieved and if needed the set to be used. Its pointers are also used to link other RECORD-MAPPINGS occurrences owned by the same RECORDS-MAPPED.

SETS-MAPPED contains the network set names used to retrieve data for a relation. Its pointers are also used to link other SETS-MAPPED occurrences owned by RELATIONS-MAPPED.

3.2.5 MAPPING OPERATORS

There are five obtain operators which can be used to navigate through the network database in the data structure

mappings and each has an associated operation code. The operators and their op-code is given in Appendix D. Figure III-5 shows how the example batch input cards for relation ABC would be translated.

```
                IDMS Operators Needed

OBTAIN NEXT A WITHIN REGION-A
OBTAIN NEXT B WITHIN A-B
OBTAIN NEXT C WITHIN A-C
END OF MAP

                Becomes Mapping Cards
6 A
2 B                         A-B
2 C                         A-C
9
```

Figure III-5. How to Build Mapping Cards

The RECORDS-MAPPED, SETS-MAPPED and ATTRIBUTE-MAPPED record types are used not only to define the relations, but also to lessen the amount of space needed to store the mapping operators. The generic mapping program, STOREMAP, and data retrieval program, NETTOREL, stores this data in a table structure and associates an index with each record type name, set type name and attribute name. These indices are then used to formulate the mapping operator along with op-codes that form an entry of up to three numbers to be translated from their equivalent character strings. An example of how these operator mappings are formulated is given in Figure III-6.

```
                    RELATION ABC

SET(1)          SET(2)
A-B             A-C


RECORD(1)       RECORD(2)       RECORD(3)
A               B               C


STORED MAPPING
6 1
2 2 1
2 3 2
9
```

Figure III-6. Data Dictionary Table

3.3 GENERIC DATA RETRIEVAL PROGRAM

The generic data retrieval program, NETTOREL, uses the data dictionary and mapping operators stored by the mapping program, STOREMAP, with the relational algebra query to generate a result relation.

3.3.1 BATCH INPUT

The first batch input card contains the name of the subschema to be used, which contains the name of the database to be queried. Only one network database may be queried under this design. All others are relational algebra query cards to and result cards which contain the name to be associated with the resulting relation. Multiple queries are allowed. Figure III-7 is an example of what the batch cards might look like for the query examples already given.

```
First query :
Retrieve relation ABC.
Second query:
Retrieve tuples in ABC where attribute B1 has a
value of b1.

          SUBSCHEMA ABC-SUB-SCHEMA
          QUERY     ABC.?
          RESULT    RELATION ABC
          QUERY     ABC. :(B1 = "b1")?
          RESULT    ABC WHERE B1 IS b1
```

Figure III-7. Example Query Batch Input Cards

Syntax checks are made on the query to insure that it conforms to the syntax given in Appendix A. To make it easier for NETTOREL to interpret the query special delimiter characters were defined. For example to signify the end of a relation name a period is used. To signify the start and end of a select parenthesis are placed around all select clause. In addition parenthesis can be used to set precedence for the logical operators and/or. The slash (/) is used to show the end of a relational operator (select, project or join) and a question mark (?) denotes the end of a query.

3.3.2 DATA STRUCTURE MAPPING

The network data structure mapping which gives the navigation path of the relation is stored within the network database. Each obtain operator has a unique op-code associated with it and an index to the record type to be obtained as well as the set type if needed.

The navigation path to be used to retrieve the data by NETTOREL is dependent on the relational algebra query. If it contains a select or join operation on an attribute which is the key field in a network record type which can be reached via an obtain calc operation then that record types mapping records will be used to navigate the database. If an obtain calc can not be used then the mappings of the first record type which contains an attribute selected or a record type in common with both relations in a join will be used. If there is no select or join clause then the mappings of the first record type are used.

3.3.3 QUERY IMPLEMENTATION

When a relation name is encountered in a query its structure information is read off of the network database data dictionary and stored within the generic program's, NETTOREL, data dictionary table in working storage. The relational select clauses cause extra tests to be performed on retrieved record occurrences to see if a record occurrence's data is to be moved to the result area or discarded. A join causes the mapping records of two relations to form a new mapping for the relation which results from the join. The project operator changes which fields are to be shifted from a record occurrence which meets selection criteria to the result area.

3.3.4 HANDLING DUPLICATES

When a tuple is created it does not have to contain data from all the record types used to navigate the database. In instances where all of a record type is not used or even none of a record type is used two path completions may differ only by the data skipped. This means duplicate tuples could be stored in the result area which is not allowed according to the rules of relational algebra.

To avoid the possibility of duplicates occurring in a resulting relation a sort technique is used to group duplicates together so they will be easy to find and eliminate.

3.3.5 OUTPUT

The output for the relational algebra query is in a table format with a maximum width of 132 characters. The attribute names are used to label each column and the width of the column is determined by the attribute width or the width of the attribute name, whichever is greater.

CHAPTER IV

SYSTEM IMPLEMENTATION

4.1 GENERAL IMPLEMENTATION

The implementation of this thesis research consists of two generic IDMS COBOL application programs called STOREMAP and NETTOREL. STOREMAP is a mapping storage program w. ich is used to fill the relational data dictionary. The information stored in the dictionary defines the relations which can be queried against by relational algebra. It also stores the mappings of the network data structure to the relational data structure. NETTOREL uses the data dictionary, data structure mappings and relational algebra query to create a result relation.

No IDMS DML statements are used in either program, STOREMAP or NETTOREL. Instead their equivalent COBOL call statements are used to enable both programs, STOREMAP and NETTOREL, to access any IDMS database. This is provided by passing variables rather then literals to the IDMS database manager.

The paragraphs within both programs were developed in a hierarchical structure for ease of maintenance and understanding. A special paragraph numbering scheme was used to represent the calling sequence of the paragraphs. For example paragraph 1.1.1 is called by paragraph 1.1. If the first character is an A, such as A.1 then the paragraph is multipurpose and is performed by several other paragraphs. A data structure chart is given in Appendix E for both STOREMAP and NETTOREL.

COBOL requires that all data storage be defined at compile time. This means that all the tables used by both generic programs are predefined and set limitations have been placed on the size of the relations which can be queried. These limitations are given in Appendix H.

4.2 GENERIC MAPPING PROGRAM

An IDMS application program STOREMAP was written in COBOL to store the mapping of the CODASYL to relational data structures within the database being queried. The IDMS operators were written as COBOL call statements instead of IDMS DML to make the STOREMAP generic.

4.2.1 DATA STRUCTURE MAPPING

Each network database which is to use the generic data retrieval program NETTOREL must have a subschema which contains all of the record types to be used in a relational query. This is provided to the generic program through a

batch input on the first card. The format of this card is provided in Appendix G.

4.2.2 NETWORK STRUCTURE

There are no limitations placed on the types of IDMS databases which may be used with these generic programs. However for a record type to be included in a relation it must be connected via a set type to at least one other record type in the relation or be the only record type in a relation. An additional requirement is that all record types in a relation must be connected via set types. Since the data structure mappings are defined by the DBA it is up to him to determine which path to define if multiple paths exists through the record types. In fact the same record types and set types can be used in different relations with different path definitions.

4.2.3 DATA DICTIONARY

The relational data dictionary consists of five record types and four set types which are to be defined with a subschema prior to executing the generic mapping program STOREMAP within each database to be queried. The STOREMAP can then be used to store the data in the data dictionary for each relation to be defined in terms of the CODASYL database. This must be done prior to executing the generic data retrieval program, NETTOREL. This is the

responsibility of the DBA since he is the most knowledgable of the structure of the database and the intent of the user.

4.2.4 MAPPING OPERATORS

A series of IDMS operators defining the relations in terms of a navigation of the network data structure is also stored on the database with the mapping program, STOREMAP. There are five possible obtain operators and they are listed in Appendix D.

To lessen the amount of storage space needed to store these operators the set types and record types were read into a table in the mapping program, STOREMAP. Their indices are used as a code along with an associated op-code for each type of obtain. Rules for how these are formulated is included in Appendix H. The STOREMAP stores the set types and record types on the data dictionary in the order of their indices so that the data retrieval program, NETTOREL, can interpret the mappings the same way the mapping program did.

4.3 GENERIC DATA RETRIEVAL PROGRAM

The generic data retrieval program, NETTOREL, uses batch input cards to set up the database being used. These cards also contain the relational algebra queries to be implemented and the names to be associated with the result relations. The syntax for the batch input cards are given in Appendix G.

4.3.1 QUERY RETRIEVAL

The relational algebra query is to be in the format of Appendix A and uses the symbols defined there. It is possible for multiple queries to be requested as long as they are all against the same database and their record types are defined within the specified subschema. The format of the query cards is listed in Appendix G.

4.3.2 DATA STRUCTURE MAPPING

The data dictionary entry for each relation is stored in the generic programs', STOREMAP and NETTOREL, working storage in the form of a table. Each record type and set type has an index associated with it depending on which record occurrence it is on the database. These indices were generated by the mapping program, STOREMAP, depending on the order of the batch input cards and the order of the previous records on the database.

4.3.3 QUERY IMPLEMENTATION

When a relation name is encountered in a query its structure information is read off the network database data dictionary and stored within NETTOREL's data dictionary table in working storage. The relation operators encountered cause flags to be set in the data dictionary table within the the table entry for the record type which contains the attribute involved in the operation. The

associated data needed for the operation is stored in a separate table for that operator.

After the entire query has been processed and the associated select, project and join flags set then NETTOREL determines which mappings to use and stores that in a separate mapping table. It then processes that table in sequential order. If a particular return causes the end of set flag to be set then the mapping index is decremented. If the retrieval is successful then select criterion are check. If they are successful then the mapping index is incremented. When the end of mapping op-code is encountered then the projects are performed and the tuple is stored in the result table in its appropriate location.

4.3.3.1 SELECT IMPLEMENTATION

After a successful data retrieval has been completed on a record type its select flag is checked to see if it is set (greater then zero). When the flag is set all attributes contained within that record is checked to see if they are part of a select clause. This is indicated by the presence of a 1 (left side) or 2 (right-side) in the select side variable.

When the variable is suppose to be on the right side of a select clause then the retrieved value of that attribute is stored in the compare field of the appropriate select table entry (the value in the select table index for that

attribute). When it is on the left side then it is compared to the value in the compare field and the results are pushed onto a select stack. Retained within the select table entry is an additional boolean operator and a disposition flag to indicate if the selection is complete. When complete the select stack is popped until an open parenenthis is found and if it is the first entry on the stack then the truth value is stored in the result-flag. If the result flag is false then the contents of the current record is not stored in the current tuple being built.

4.3.3.1 PROJECT IMPLEMENTATION

When the end of mapping op-code is encountered then the project table is checked to see which of the result fields are to be stored in the final tuple result. The result table consists of the result working area displacements and field lengths. These are used with a series of move statements to create the final tuple. The tuple is then placed in its appropriate location in the result table.

4.3.3.3 JOIN IMPLEMENTATIONS

The join operator is used to create the mapping table to be used to navigate the database. The record type to be joined over will determine which mappings from each of the relations being joined is to be used. The mapping associated with that record type in the first relation will

be used in the first part of the mapping table and the mapping associated with that record type in the second relation will follow all the mappings in the first relation with the end of map mapping removed and the first mapping retrieving the common record type removed from the second set of mappings.

4.3.3 HANDLING DUPLICATES

A check must be made within the data retrieval program, NETTOREL, to avoid storing duplicate tuples. This could have been done by sorting the result file after it was completely filled and removing duplicates within the sort. This would allow all of the result information to be maintained on secondary storage. It also would take longer to implement a routine to handle an optimal sort then time permitted within this thesis effort and the issue is not pertinent to the objective of the effort.

The approach that was taken was to keep the result file in a table form within working storage. As a tuple is created it is added in the table in the appropriate sorted order. This is done by comparing the new tuple to the current last tuple. If it is greater, it is added in the next table location. If it isn't, then it is compared to the next, until a tuple is found of equal or lesser value. This indicates that the tuple is to be rejected or inserted

following the lesser tuple. This will take advantage of databases which are stored in an ascending sorted order.

4.3.4 OUTPUT

Each time a path is completed the resulting tuple will be stored in the result table as specified above. After all tuples have been retrieved the result table will be written to the specified output file. The size of the record will be limited to 132 characters and its format will be determined by the width of the attribute name or size depending on which is longer.

CHAPTER V

TESTING

5.1 TESTING DATABASE DESCRIPTION

An IDMS demonstration customer database set up by the Cullinet Corporation was used to test both generic programs, STOREMAP and NETTOREL. It consists of four areas (CUSTOMER-REGION, ORDER-REGION, PRODUCT-REGION, AND SALES-REGION) and eight record types (CUSTOMER, ORDOR, ITEM, OREMARK, PRODUCT, SALES, CTRL AND CORPIND). The structure of the database is given in Figure V-1. The relation structures is given in FIgure V-2. The testing plan for both generic programs is included in Appendix K.

5.2 GENERIC MAPPING PROGRAM

The testing database chosen did not provide a through validation test of the generic mapping program STOREMAP. To do so would require testing the program with several different IDMS databases with varying levels of complexity. The possible data structure of a network database are numerous and to test all of them would be too time consuming for this thesis effort.

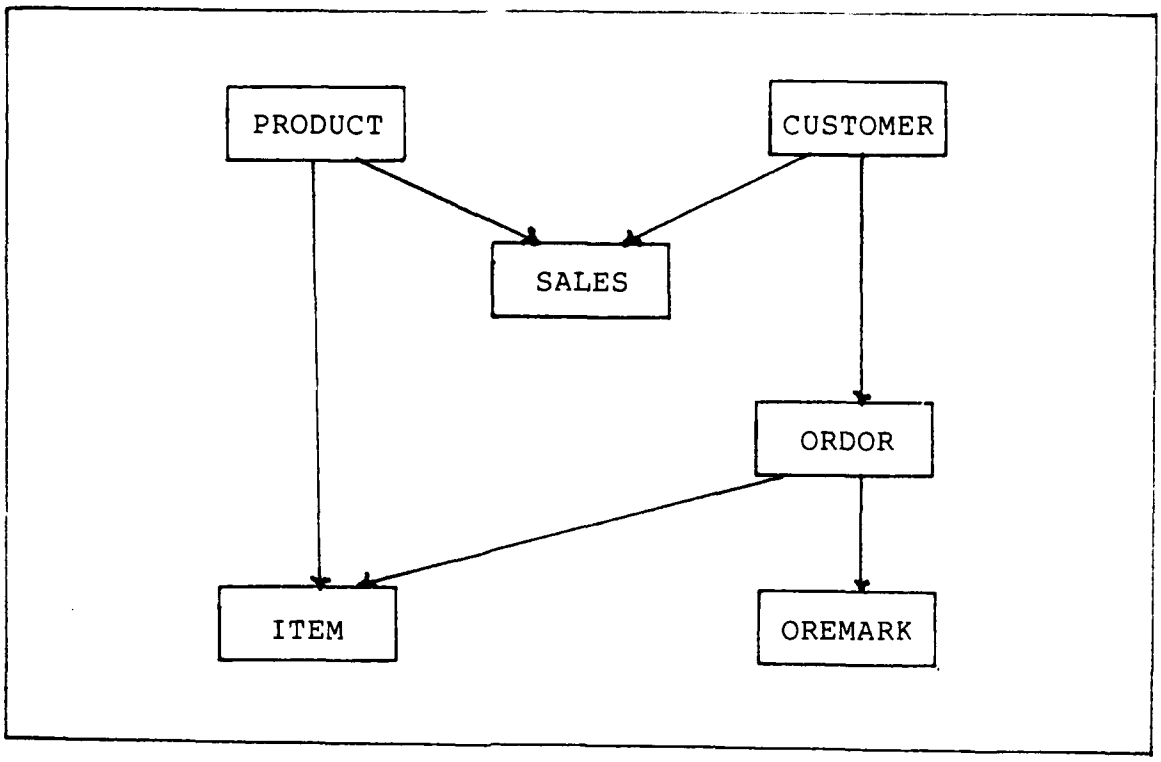


Figure V-1. Test Database Bachman Diagrams

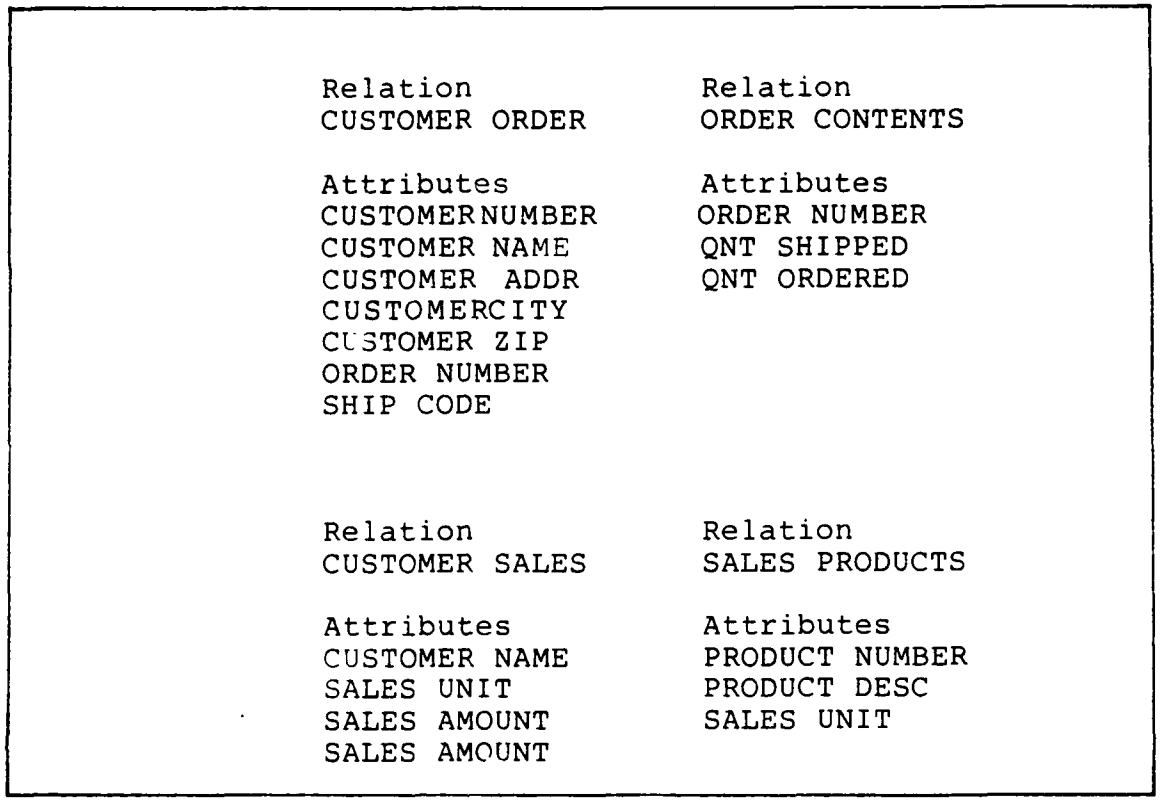


Figure V-2. Test Relation Structure

5.2.1 TESTED DATA STRUCTURES

The tested database contained a looping structure which made it possible to test the different mappings for each record type to ensure that the same relation was returned independent of the starting record type. It also contained record types which were owned by more than one record type and record types which owned more than one record type. In addition record types which belonged in one area owned record types in another. These are the most common data structures currently found in network databases.

5.2.2 STRUCTURES NOT TESTED

The database tested was chosen because it was already available and time did not permit the writing of the application programs needed to create and fill a new database which might contain more complicated network structures. This was left for a follow-on effort which should contain record types with multiple sets defined between them, and relations that contain link records. The Presidents database is currently being used for testing of relational database model efforts and could be set up as a network for testing this thesis effort.

5.3 GENERIC DATA RETRIEVAL PROGRAM

The generic data retrieval program, NETTOREL, did not receive a through test because a variety of data structures were not defined as relations. But all possible

combinations of the select, project and join operators were tested. There were two types of structures for testing the join. Two relations were joined across a record type which was owned by a different record type in each relation and two relations were joined over a record type which owned a different record type in each relation.

5.4 SYNTAX CHECKS

Both programs, STOREMAP and NETTOREL, used a batch input which required that they perform syntax checks on the cards read in. The syntax checks were set up according to Appendix A and G. The test plan incorporated tests to verify that syntax errors were detected.

CHAPTER VI

CONCLUSIONS AND RECOMMENDATIONS

6.1 CONCLUSIONS

The basis to this thesis effort was to use a network application program to provide a relational view of a network database. This was done by storing information about the record and set types used in each relation in a relational data dictionary and a set of network operators to define the mapping between a network data structure and a relational data structure. This information was then used by a generic data retrieval program, NETTOREL, along with a relational algebra query to create a relational result table.

6.1.1 NETWORK IMPROVEMENTS

The network model is more efficient because it has a great deal of control over how the data is stored and accessed. This control also forces the user to have a strong grasp of the network data structure to navigate the database. In fact most user's have to rely on an

application programmer to write an interface for them to access their data.

This thesis project still needs to know how to navigate the database, but once that is defined, by the DBA, for each relation to be queried, the users can write their own relational algebra queries. This lessens the amount of application programs needed for different views of the database. It also allows multiple user's access to the same relations even if the data they need is very different.

6.1.2 RELATIONAL IMPROVEMENTS

The relational model is more user friendly then the network database model but because it is at so high a level of abstraction from the data storage and access it allows for implementations which have very slow data access and redundant storage of data. By using a network data structure to retrieve the data and a network application program to provide the user with a relational view of the data the efficiency of the network model can be used without giving up the ease of use of the relational model.

6.2 RECOMMENDATIONS

This thesis was a first attempt at using the network model to act as a data access interface for the relational model. There are improvements that can be made to make the system more user friendly. Both generic programs, STOREMAP

and NETTOREL would have been easier to use if an interactive input had been chosen. Not all the relational operators were implemented. In addition, no update capability has been included.

6.2.1 MAPPING PROGRAM ENHANCEMENTS

The mapping program lends itself to a menu driven data entry system. Since the record and set types available are already defined within the IDMS database this information can be retrieved from the IDMS data dictionary and displayed to the user for them to choose those record and set types to be defined within a relation.

It would also be possible for software to be generated that would take one record type relational data structure path and translate it into all the other paths needed for each record type. This would make the DBA's job easier.

6.2.2 DATA RETRIEVAL PROGRAM ENHANCEMENTS

Only the select, project and join relational algebra operators were implemented. To make the system a complete relational database model the union, intersection, difference, division and cartesian product operators should be implemented. An update capability would also be desirable in addition to some algebraic capabilities such as sums, and averages.

To allow a cartesian product operation to be perform the definition of a relation in this effort would have to be

modified since there would be no explicit relationship defined between the two relations (set type). The result relation would have essentially two separate mappings (Figure VI-1).

To provide a relational insertion and deletion capability the problem of incomplete information has to be addressed. For example, a record type might be used to navigate the database for defining a tuple for a given relation, but none of its data used. If a tuple was to be inserted the database would not have the missing data or be able to establish the pointers needed in the linking record type. Deletions could result in records being left which weren't needed or reachable and record occurrences linked to record occurrences other then the one being deleted could be lost.

Entering the relational queries interactively with a format similar to System R which uses blank tables would be more user friendly along with a interactive output in the same format.

This research also lends itself to using a natural language translator for the relational query inputs as opposed to using relational algebra queries since the relational database model has such a high level of abstraction. A translator would increase productivity and and ease of use.

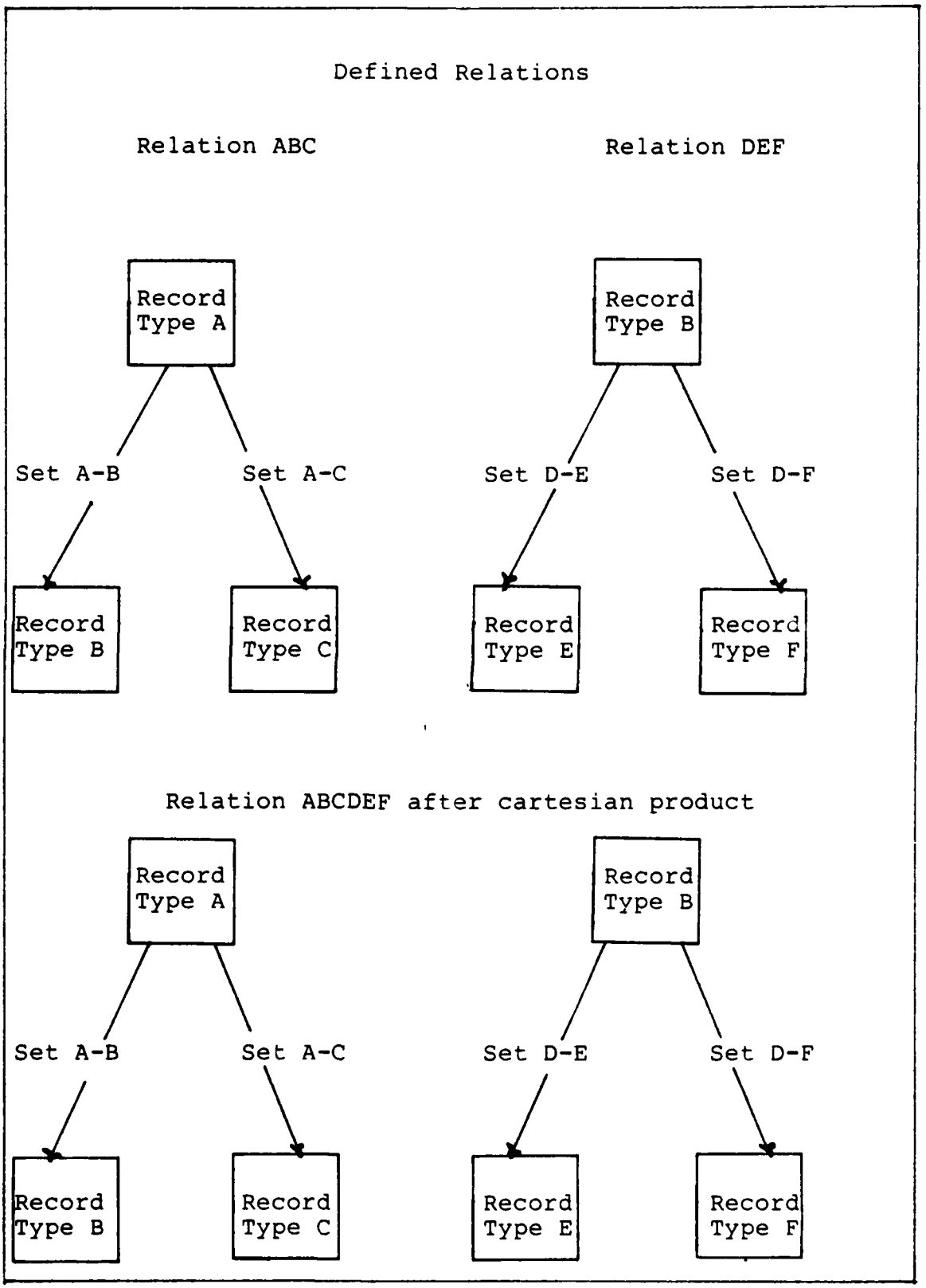


Figure VI-1. Result of a Cartesian Product

Bibliography

Aho, A. V., Y. Sagiv and J. D. Ullman, "Efficient Optimization of a Class of Relational Expressions," ACM Transactions on Database Systems:433-454,December 1979.

Babb, E. "Implementing a Relational Database by Means of Specialized Hardware", ACM Transactions on Database Systems:1-29, March 1979.

Codd, E. F., "A Relational Model for Large Shared Data Bases", Communications on the ACM: 377-387, June 1970.

----------, "Extending the Database Relational Model to Capture More Meaning", ACM Transactions on Database Systems:397-434, December 1979.

----------, "Relational Database: A Practical Foundation for Productivity", Comunications of the ACM:109-117, February 1982.

"Data Base Task Group of CODASYL Programming Language Committee Report", April 1971.

Date C. J., "An Introduction to Database Systems" Third Edition, February 1982.

Fallahzadeh H., "Design of a Relational View and Relational Operator Translator over Network Structured Databases", May 1982.

Landon, Glen G. Jr., "A Note on Associative Processors for Data Management", ACM Transactions on Database Systems, 148-158 June 1978.

Lien, Y. E., "On the Equivalence of Database Models", Journal of the ACM:33-362, April 1982.

Lin, C. S., D. C. P. Smith, and J. M. Smith, "The Design of a Rotating Associative Memory for Relational Database Application", ACM Transactions on Database Systems:53-65, March 1976.

Ozkarahan, Esen A., "Hardware and Software Systems Research on the RAP Database Machine", ICCC IEEE 894-898 1980.

Ullman Jeffrey D., "Principles of Database Systems" 1982.

APPENDIX A

RELATIONAL ALGEBRA SYNTAX

OPERATOR	SYNTAX
Select	relation. : (field name =, <, >, not) or, and (...)
Project	relation. % field name, field name, ...
Join	relation. * relation.

EXAMPLES

Find tuples where a1 = a2 and a3 <= "B" in relation A

A. : (a1 = a2 and a3 <= "B")?

Project a2, a3 and a4 from relation A

A. % a2, a3, a4?

Join relations A and B

A. * B.?

All three in one query

A. : (a1 = a2 and a3 <= "B")/ % a2, a3, a4/ * B.?

APPENDIX B

SCHEMA

```
SCHEMA NAME IS RSCHEMA      VERISON IS 1.
INSTALLATION.               ASD.

FILE DESCRIPTION.
FILE NAME IS RELATION       ASSIGN TO RELATION
                            DEVICE TYPE IS 3330B.

AREA DESCRIPTION.
AREA NAME IS RELATION-REGION
     RANGE IS 4501 THRU 5000
                                WITHIN FILE RELATION
                                       FROM 1 THRU 500.

RECORD DESCRIPTION.

RECORD NAME IS RELATIONS-MAPPED.
RECORD ID  IS 650.
LOCATION MODE IS CALC       USING RELATION-NAME
                            DUPLICATES ARE NOT ALLOWED.


WITHIN RELATION-REGION AREA.
     03   RELATION-NAME                          PIC X(16).

RECORD NAME IS RECORDS-MAPPED.
RECORD ID IS 651.
LOCATION MODE IS VIA RELATION-RECORDS SET.

WITHIN RELATION-REGION AREA.
     03   RECORD-NAMES                           PIC X(16).
     03   AREA-NAME                              PIC X(16).
     03   CALC-FLAG                              PIC 9.

RECORD NAME IS ATTRIBUTE-MAPPED.
RECORD ID IS 652.
LOCATION MODE IS VIA RECORD-ATTRIBUTE SET.

WITHIN RELATION-REGION AREA.
     03   ATTRIBUTE-NAME                         PIC X(16).
     03   ATTRIBUTE-DISP                         PIC 99.
     03  FIELD-DISP                              PIC 99.
     03   FIELD-LENGTH                           PIC 99.
     03   FIELD-TYPE.
          05   SIGN-FLAG                         PIC X.
```

```
          05   WHOLE-NUM                          PIC 9.
          05   DECIMAL-FLAG                       PIC X.
          05   DECIMAL                            PIC 9.
          05   COMP-TYPE                          PIC X.

RECORD NAME IS RECORD-MAPPINGS.
RECORD ID IS 652.
LOCATION MODE IS VIA RECORDS-MAPPING SET.

WITHIN RELATION-REGION AREA.
     03   OP-CODE                                PIC 9.
     03   RECORD-NAMES                           PIC X(16).
     03   SET-OR-AREA                            PIC 99.

RECORD NAME IS SETS-MAPPED.
RECORD ID IS 654.
LOCATION MODE IS VIA RELATIONS-SETS SET.

WITHIN RELATION-REGION AREA.
     03   SET-NAME                               PIC X(16).

SET DESCRIPTION.

SET NAME IS RELATION-RECORDS.
ORDER IS NEXT.
MODE IS CHAIN.
OWNER IS RELATIONS-MAPPED      NEXT DBKEY POSITION IS 1.
MEMBER IS RECORDS-MAPPED       NEXT DBKEY POSITION IS .
                               MANDATORY AUTOMATIC.

SET NAME IS RECORD-ATTRIBUTE.
ORDER IS NEXT.
MODE IS CHAIN.
OWNER IS RECORD-MAPPED         NEXT DBKEY POSITION IS 2.
MEMBER IS ATTRIBUTE-MAPPED     NEXT DBKEY POSITION IS 1
                               MANDATORY AUTOMATIC.

SET NAME IS RECORDS-MAPPING.
ORDER IS NEXT.
MODE IS CHAIN.
OWNER IS RECORDS-MAPPED        NEXT DBKEY POSITION IS 3.
MEMBER IS RECORD-MAPPINGS      NEXT DBKEY POSITION IS 1
                               MANDATORY AUTOMATIC.


SET NAME IS RELATION-SETS
ORDER IS NEXT.
MODE IS CHAIN.
OWNER IS RELATIONS-MAPPED      NEXT DBKEY POSITION IS 2.
MEMBER IS SETS-MAPPED          NEXT DBKEY POSITION IS 1
                               MANDATORY AUTOMATIC.
```

APPENDIX C

SUBSCHEMA

```
ADD SUBSCHEMA NAME IS RSUBSCHM
    OF SCHEMA NAME IS RSCHEMA
    DMCL NAME IS RDMCL.

ADD RECORD RELATIONS-MAPPED
    ERASE NOT ALLOWED.

ADD RECORD RECORDS-MAPPED
    ERASE NOT ALLOWED.

ADD RECORD ATTRIBUTE-MAPPED
    ERASE NOT ALLOWED.

ADD RECORD RECORD-MAPPINGS
    ERASE NOT ALLOWED.

ADD RECORD SETS-MAPPED
    ERASE NOT ALLOWED.
```

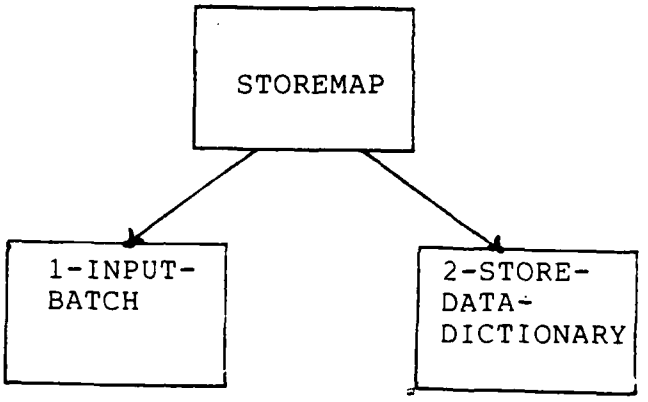
APPENDIX D

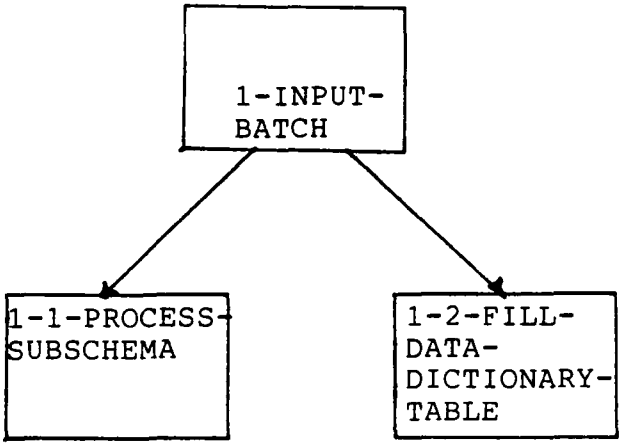
DATA STRUCTURE MAPPING OPERATORS

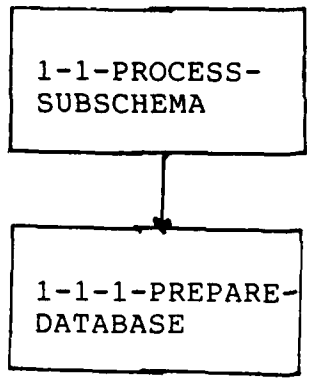
1	OBTAIN OWNER WITHIN SET
2	OBTAIN NEXT WITHIN SET
4	OBTAIN PRIOR WITHIN SET
5	OBTAIN FIRST WITHIN AREA
6	OBTAIN NEXT WITHIN AREA
8	OBTAIN PRIOR WITHIN AREA
9	END OF MAP

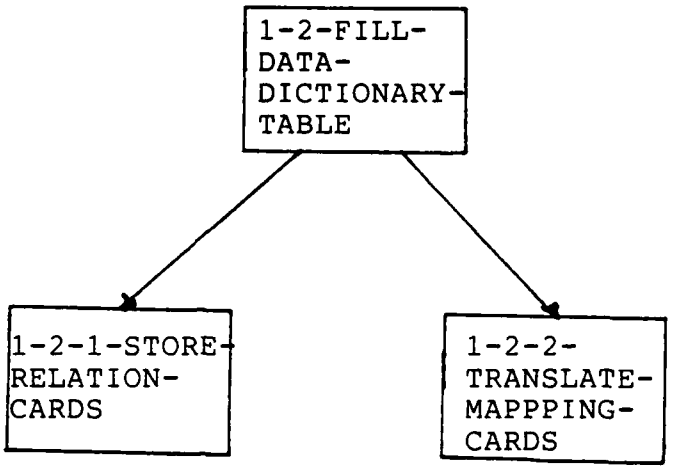
APPENDIX E

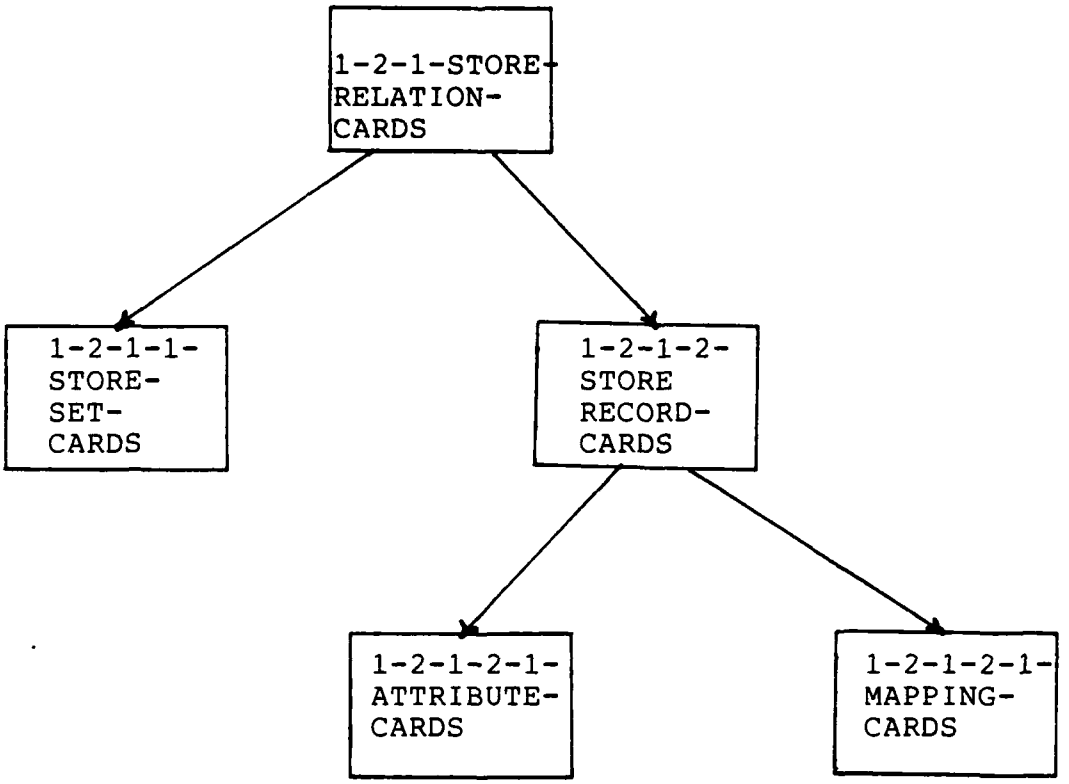
DATA STRUCTURE CHARTS

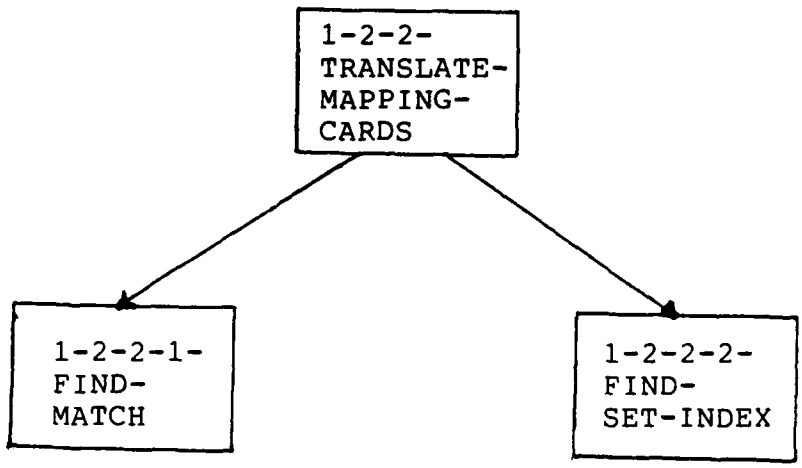


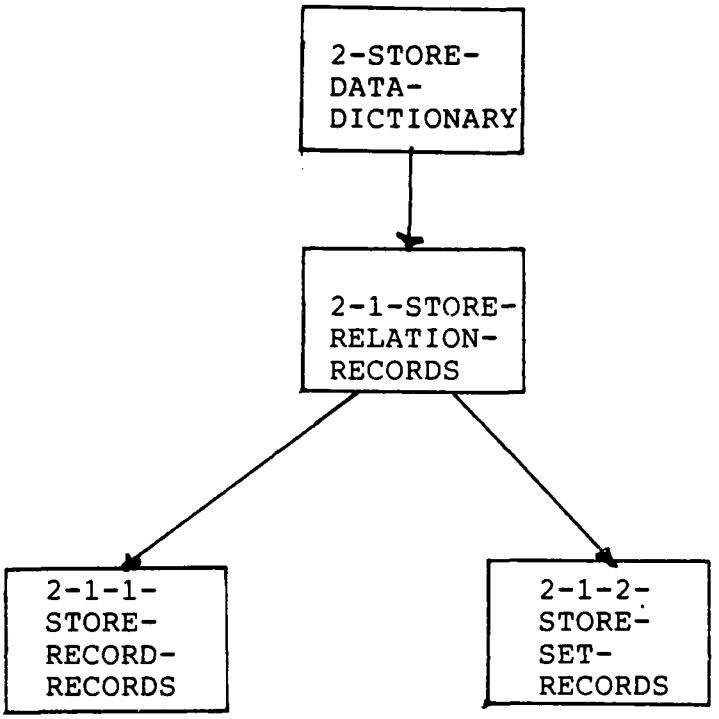

1-INPUT-
BATCH
1-1-PROCESS-
SUBSCHEMA
1-2-FILL-
DATA-
DICTIONARY-
TABLE

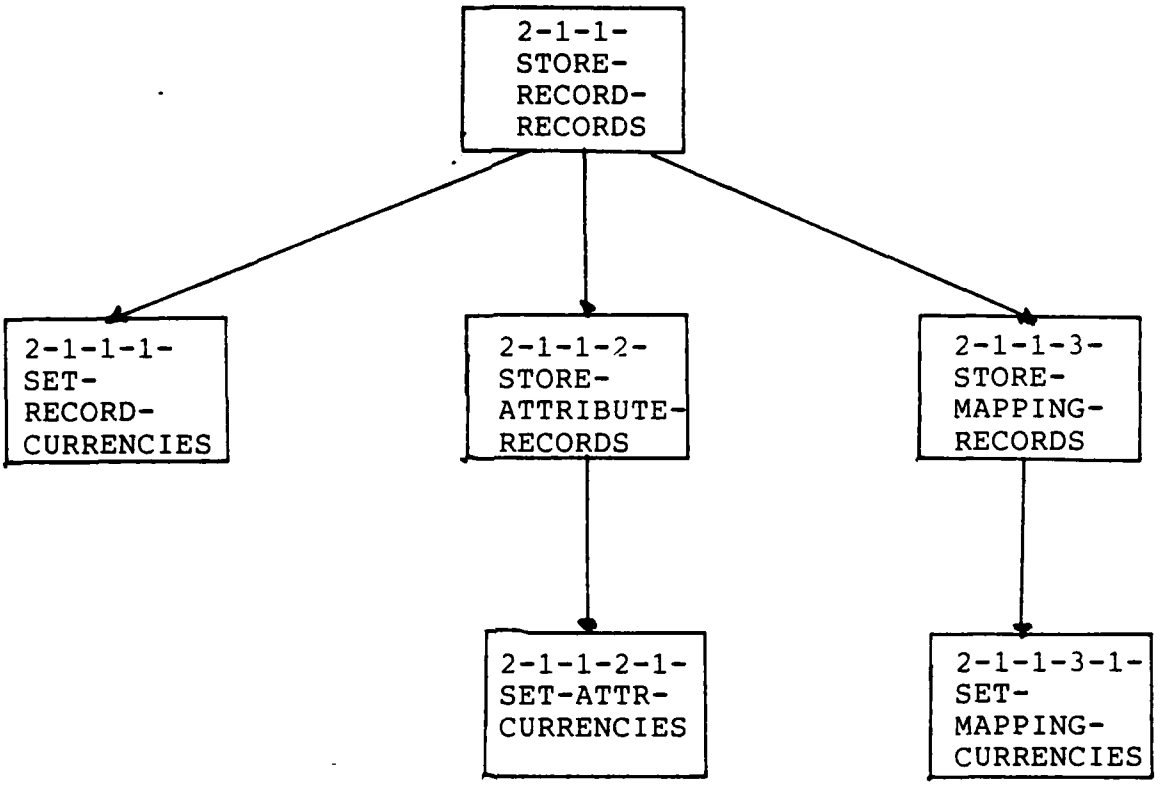

1-1-PROCESS-
SUBSCHEMA
1-1-1-PREPARE-
DATABASE

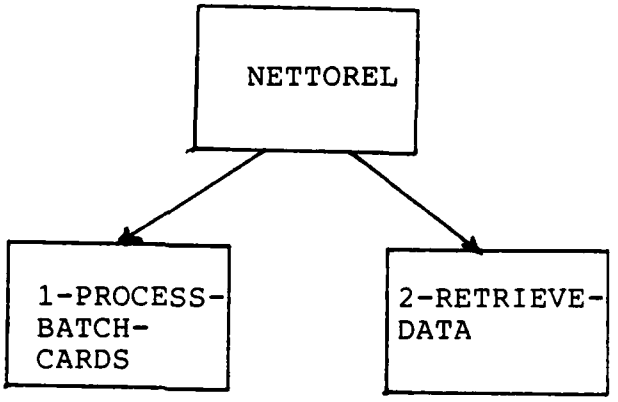

1-2-FILL-
DATA-
DICTIONARY-
TABLE
1-2-1-STORE-
RELATION-
CARDS
1-2-2-
TRANSLATE-
MAPPPING-
CARDS

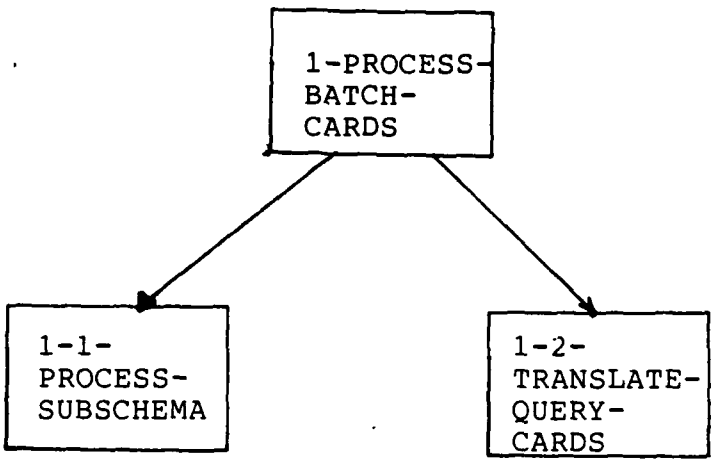

1-2-1-STORE-RELATION-CARDS
1-2-1-1-STORE-SET-CARDS
1-2-1-2-STORE RECORD-CARDS
1-2-1-2-1-ATTRIBUTE-CARDS
1-2-1-2-1-MAPPING-CARDS

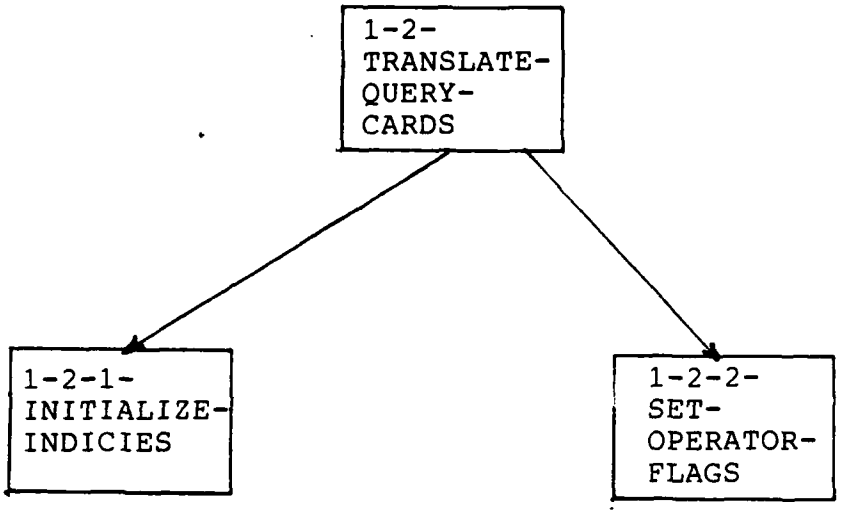

1-2-2-
TRANSLATE-
MAPPING-
CARDS
1-2-2-1-
FIND-
MATCH
1-2-2-2-
FIND-
SET-INDEX

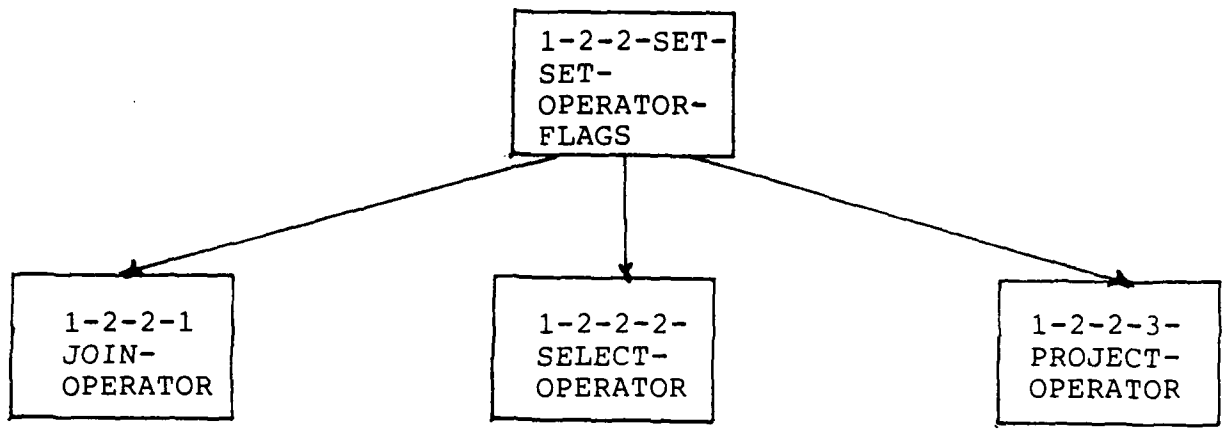

2-STORE-
DATA-
DICTIONARY
2-1-STORE-
RELATION-
RECORDS
2-1-1-
STORE-
RECORD-
RECORDS
2-1-2-
STORE-
SET-
RECORDS

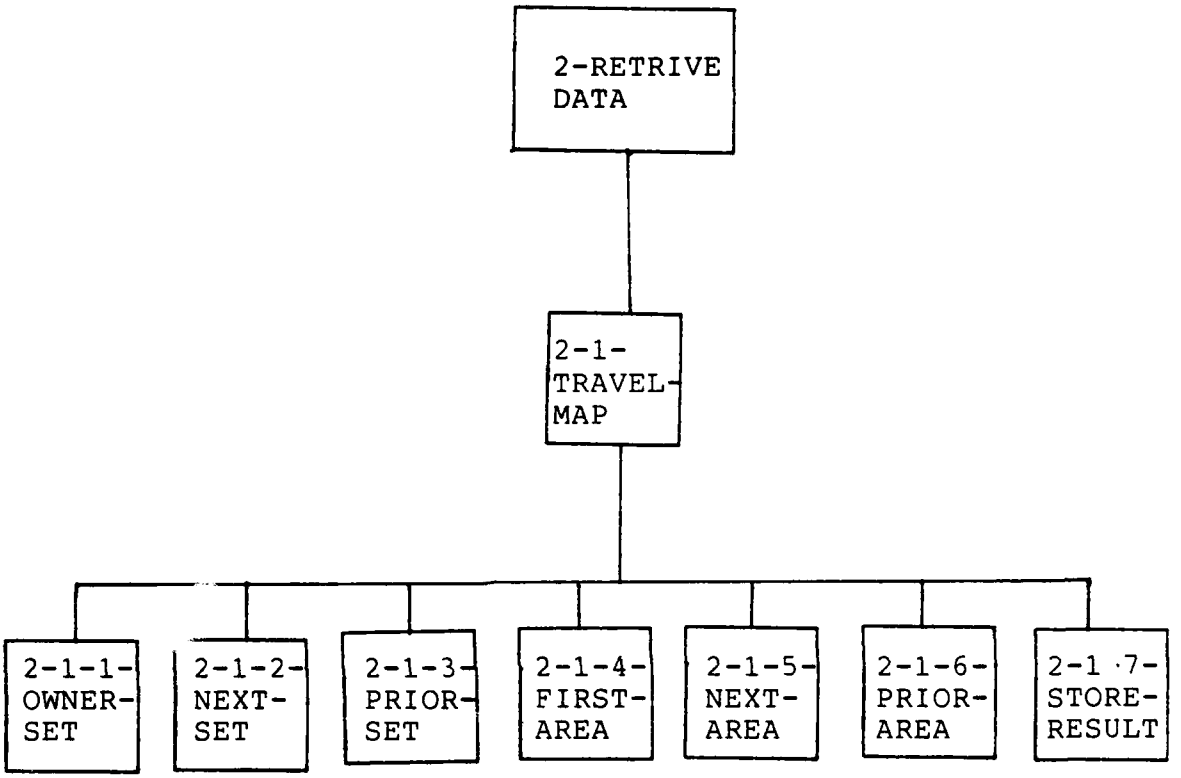

2-1-1-
STORE-
RECORD-
RECORDS
2-1-1-1-
SET-
RECORD-
CURRENCIES
2-1-1-2-
STORE-
ATTRIBUTE-
RECORDS
2-1-1-3-
STORE-
MAPPING-
RECORDS
2-1-1-2-1-
SET-ATTR-
CURRENCIES
2-1-1-3-1-
SET-
MAPPING-
CURRENCIES


NETTOREL
1-PROCESS-BATCH-CARDS
2-RETRIEVE-DATA


1-PROCESS-
BATCH-
CARDS
1-1-
PROCESS-
SUBSCHEMA
1-2-
TRANSLATE-
QUERY-
CARDS


1-2-
TRANSLATE-
QUERY-
CARDS
1-2-1-
INITIALIZE-
INDICIES
1-2-2-
SET-
OPERATOR-
FLAGS


1-2-2-SET-SET-OPERATOR-FLAGS
1-2-2-1 JOIN-OPERATOR
1-2-2-2-SELECT-OPERATOR
1-2-2-3-PROJECT-OPERATOR

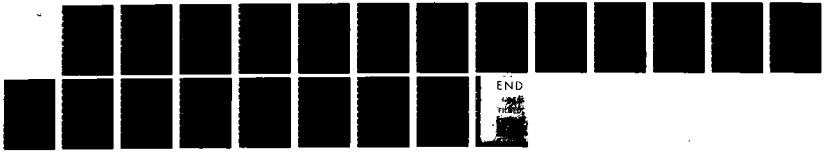

2-RETRIVE DATA
2-1-TRAVEL-MAP
2-1-1-OWNER-SET
2-1-2-NEXT-SET
2-1-3-PRIOR-SET
2-1-4-FIRST-AREA
2-1-5-NEXT-AREA
2-1-6-PRIOR-AREA
2-1-7-STORE-RESULT

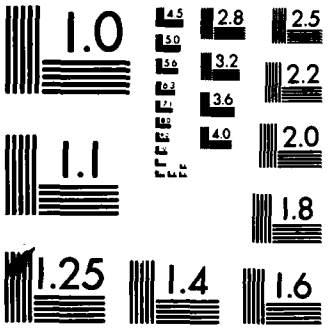
APPENDIX F

PAGE NO. 00001 NETTOREL Data Dictionary

LEVEL	VARIABLE NAME	PART OF VARIABLE
1	INPUT-CARD	
3	CARD-TYPE	INPUT-CARD
3	TYPE-DATA	INPUT-CARD
3	SCHEMA-DATA	INPUT-CARD
5	SUBSCHEMA-NAME	SUBSCHEMA-DATA
3	QUERY-DATA	INPUT-CARD
5	QUERY-CHARACTERS	QUERY-DATA
7	QUERY-STRING	QUERY-CHARACTERS
3	RESULT-FILE	INPUT-CARD
5	RESULT-TABLE-NAME	RESULT-FILE
3	ERROR-DATA	INPUT-CARD
5	ERROR-MESSAGE	ERROR-DATA
5	ERROR-CHARACTER	ERROR-DATA
5	ERROR-LOCATION	ERROR-DATA
1	RELATION-REPORT-TABLE	
3	REPORT-LINE	RELATION-REPORT-TABLE
1	SELECT-TABLE	
3	MAX-SELECTS	SELECT-TABLE
3	RECORD-SELECTS	SELECT-TABLE
5	LEFT-VARIABLE	RECORD-SELECTS

AD-A138 117 THE OPERATOR MAPPING BETWEEN RELATIONAL ALGEBRA 2/2
OPERATORS AND CODASYL BAS..(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. D J NICELY
UNCLASSIFIED DEC 83 AFIT/GCS/EE/83D-15 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

PAGE NO. 00002 NETTOREL Data Dictionary

LEVEL	VARIABLE NAME	PART OF VARIABLE
5	SELECTOR-CHARACTER	RECORD-SELECTS
5	COMPARE-FIELD	RECORD-SELECTS
5	RIGHT-VARIABLE	RECORD-SELECTS
5	AND-OR-FLAG	RECORD-SELECTS
5	OPERATOR-DISPOSITION	RECORD-SELECTS
1	PROJECT-TABLE	
3	MAX-PROJECTS	PROJECT-TABLE
3	ATTRIBUTE-PROJECTED	PROJECT-TABLE
5	PROJECT-RELATION	ATRIBUTE-PROJECTED
5	PROJECT-RECORD	ATTRIBUTE-PROJECTED
5	PROJECT-ATTRIBUTE	ATTRIBUTE-PROJECTED
1	JOIN-TABLE	
3	MAX-JOINS	JOIN-TABLE
3	RELATIONS-JOINED	JOIN-TABLE
5	MEMBER	RELATIONS-JOINED
7	MEMBER-NAME	MEMBER
1	DATA-DICTIONARY-TABLE	
3	MAX-RELATIONS	DATA-DICTIONARY-TABLE
3	RELATIONS	DATA-DICTIONARY-TABLE
5	RELATION-NAME	RELATIONS

PAGE NO. 00003 NETTOREL Data Dictionary

LEVEL	VARIABLE NAME	PART OF VARIABLE
5	MAX-SETS	RELATIONS
5	SETS	RELATIONS
7	SET-NAME	SETS
5	MAX-RECORDS	RELATIONS
5	RECORD-S	RELATIONS
7	RECORD-NAMES	RECORD-S
7	AREA-NAMES	RECORD-S
7	CALC-FLAG	RECORD-S
7	SELECT-FLAG	RECORD-S
7	JOIN-FLAG	RECORD-S
7	MAX-ATTRIBUTES	RECORD-S
7	ATTRIBUTES	RECORD-S
9	ATTRIBUTE-NAME	ATTRIBUTES
9	ATTRIBUTE-DISP	ATTRIBUTES
9	FIELD-DISP	ATTRIBUTES
9	FIELD-LENGTH	ATTRIBUTES
9	FIELD-TYPE	ATTRIBUTES
11	SIGN-FLAG	FIELD-TYPE
11	WHOLE-NUM	FIELD-TYPE
11	DECIMAL-FLAG	FIELD-TYPE

PAGE NO. 00004 NETTOREL Data Dictionary

LEVEL	VARIABLE NAME	PART OF VARIABLE
11	DECIMALS	FIELD-TYPE
11	COMP-TYPE	FIELD-TYPE
9	SELECT-SIDE	ATTRIBUTES
9	PROJECT-FLAG	ATTRIBUTES
9	SELECT-TABLE-INDEX	ATTRIBUTES
7	MAX-MAPPINGS	RECORD-S
7	MAPPINGS	RECORD-S
9	OP-CODE	MAPPINGS
9	RECORDS-INDEX	MAPPINGS
9	SETS-INDEX	MAPPINGS
1	HOLDING-AREA	
3	WHOLE-STRING	HOLDING-AREA
3	SINGLE-CHARACTERS	HOLDING-AREA
5	EACH-CHARACTER	SINGLE-CHARACTERS
7	HOLDING-STRING	EACH-CHARACTER
3	QUERY	HOLDING-AREA
5	QUERY-PART	QUERY
7	IMEDIATE-VALUE	QUERY-PART
9	FIRST-CHARACTER	IMEDIATE-VALUE
9	IMEDIATE-STRING	IMEDIATE-VALUE

PAGE NO. 00005 NETTOREL Data Dictionary

LEVEL	VARIABLE NAME	PART OF VARIABLE
5	FIND-NAMES	QUERY
7	RELATION-NAME	FIND-NAMES
7	ATTRIBUTE-NAME	FIND-NAMES
5	FIND-SELECTOR	QUERY
7	SELECTOR	FIND-SELECTOR
7	FIND-NOT	FIND-SELECTOR
5	FIND-LOGICAL	LOGICL-OP
7	LOGICAL-OP	FIND-LOGICAL
7	QUERY-OPERATOR	FIND-LOGICAL
1	MAPPING-TABLE	
3	MAPPING-TABLE-ENTRY	MAPPING-TABLE
5	MAPPED-OP-CODE	MAPPING-TABLE-ENTRY
5	MAPPED-RELATIONS	MAPPING-TABLE-ENTRY
5	MAPPED-RECORDS	MAPPING-TABLE-ENTRY
5	MAPPED-SETS	MAPPING-TABLE-ENTRY
1	SETS-MAPPED	
3	SET-NAME	SETS-MAPPED
1	RECORD-MAPPINGS	
3	OP-CODE	RECORD-MAPPINGS
3	RECORD-NAMES	RECORD-MAPPINGS

PAGE NO. 00006 NETTOREL Data Dictionary

LEVEL	VARIABLE NAME	PART OF VARIABLE
3	SET-OR-AREA	RECORD-MAPPINGS
1	ATTRIBUTE-MAPPED	
3	ATTRIBUTE-NAME	ATTRIBUTE-MAPPED
3	ATTRIBUTE-DISP	ATTRIBUTE-MAPPED
3	FIELD-DISP	ATTRIBUTE-MAPPED
3	FIELD-LENGTH	ATTRIBUTE-MAPPED
3	FIELD-TYPE	ATTRIBUTE-MAPPED
1	RECORDS-MAPPED	
3	RECORD-NAMES	RECORDS-MAPPED
3	AREA-NAMES	RECORDS-MAPPED
3	CALC-FLAG	RECORDS-MAPPED
1	RELATIONS-MAPPED	
3	RELATION-NAME	RELATIONS-MAPPED

PAGE NO. 00001 STOREMAP Data Dictionary

LEVEL	VARIABLE NAME	PART OF VARIABLE
1	INPUT-CARD	
3	CARD-TYPE	INPUT-CARD
3	TYPE-DATA	INPUT-CARD
3	SCHEMA-DATA	INPUT-CARD
5	SUBSCHEMA-NAME	SUBSCHEMA-DATA
3	QUERY-DATA	INPUT-CARD
5	QUERY-CHARACTERS	QUERY-DATA
7	QUERY-STRING	QUERY-CHARACTERS
3	RESULT-FILE	INPUT-CARD
5	RESULT-TABLE-NAME	RESULT-FILE
3	ERROR-DATA	INPUT-CARD
5	ERROR-MESSAGE	ERROR-DATA
5	ERROR-CHARACTER	ERROR-DATA
5	ERROR-LOCATION	ERROR-DATA
1	RELATION-REPORT-TABLE	
3	REPORT-LINE	RELATION-REPORT-TABLE
1	SELECT-TABLE	
3	MAX-SELECTS	SELECT-TABLE
3	RECORD-SELECTS	SELECT-TABLE
5	LEFT-VARIABLE	RECORD-SELECTS

PAGE NO. 00002 STOREMAP Data Dictionary

LEVEL	VARIABLE NAME	PART OF VARIABLE
5	SELECTOR-CHARACTER	RECORD-SELECTS
5	COMPARE-FIELD	RECORD-SELECTS
5	RIGHT-VARIABLE	RECORD-SELECTS
5	AND-OR-FLAG	RECORD-SELECTS
5	OPERATOR-DISPOSITION	RECORD-SELECTS
1	PROJECT-TABLE	
3	MAX-PROJECTS	PROJECT-TABLE
3	ATTRIBUTE-PROJECTED	PROJECT-TABLE
5	PROJECT-RELATION	ATRIBUTE-PROJECTED
5	PROJECT-RECORD	ATTRIBUTE-PROJECTED
5	PROJECT-ATTRIBUTE	ATTRIBUTE-PROJECTED
1	JOIN-TABLE	
3	MAX-JOINS	JOIN-TABLE
3	RELATIONS-JOINED	JOIN-TABLE
5	MEMBER	RELATIONS-JOINED
7	MEMBER-NAME	MEMBER
1	DATA-DICTIONARY-TABLE	
3	MAX-RELATIONS	DATA-DICTIONARY-TABLE
3	RELATIONS	DATA-DICTIONARY-TABLE
5	RELATION-NAME	RELATIONS

PAGE NO. 00003 STOREMAP Data Dictionary

LEVEL	VARIABLE NAME	PART OF VARIABLE
5	MAX-SETS	RELATIONS
5	SETS	RELATIONS
7	SET-NAME	SETS
5	MAX-RECORDS	RELATIONS
5	RECORD-S	RELATIONS
7	RECORD-NAMES	RECORD-S
7	AREA-NAMES	RECORD-S
7	CALC-FLAG	RECORD-S
7	SELECT-FLAG	RECORD-S
7	JOIN-FLAG	RECORD-S
7	MAX-ATTRIBUTES	RECORD-S
7	ATTRIBUTES	RECORD-S
9	ATTRIBUTE-NAME	ATTRIBUTES
9	ATTRIBUTE-DISP	ATTRIBUTES
9	FIELD-DISP	ATTRIBUTES
9	FIELD-LENGTH	ATTRIBUTES
9	FIELD-TYPE	ATTRIBUTES
11	SIGN-FLAG	FIELD-TYPE
11	WHOLE-NUM	FIELD-TYPE
11	DECIMAL-FLAG	FIELD-TYPE

PAGE NO. 00004 STOREMAP Data Dictionary

LEVEL	VARIABLE NAME	PART OF VARIABLE
11	DECIMALS	FIELD-TYPE
11	COMP-TYPE	FIELD-TYPE
9	SELECT-SIDE	ATTRIBUTES
9	PROJECT-FLAG	ATTRIBUTES
9	SELECT-TABLE-INDEX	ATTRIBUTES
7	MAX-MAPPINGS	RECORD-S
7	MAPPINGS	RECORD-S
9	OP-CODE	MAPPINGS
9	RECORDS-INDEX	MAPPINGS
9	SETS-INDEX	MAPPINGS
1	HOLDING-AREA	
3	WHOLE-STRING	HOLDING-AREA
3	SINGLE-CHARACTERS	HOLDING-AREA
5	EACH-CHARACTER	SINGLE-CHARACTERS
7	HOLDING-STRING	EACH-CHARACTER
3	QUERY	HOLDING-AREA
5	QUERY-PART	QUERY
7	IMEDIATE-VALUE	QUERY-PART
9	FIRST-CHARACTER	IMEDIATE-VALUE
9	IMEDIATE-STRING	IMEDIATE-VALUE

PAGE NO. 00005 STOREMAP Data Dictionary

LEVEL	VARIABLE NAME	PART OF VARIABLE
5	FIND-NAMES	QUERY
7	RELATION-NAME	FIND-NAMES
7	ATTRIBUTE-NAME	FIND-NAMES
5	FIND-SELECTOR	QUERY
7	SELECTOR	FIND-SELECTOR
7	FIND-NOT	FIND-SELECTOR
5	FIND-LOGICAL	LOGICL-OP
7	LOGICAL-OP	FIND-LOGICAL
7	QUERY-OPERATOR	FIND-LOGICAL
1	MAPPING-TABLE	
3	MAPPING-TABLE-ENTRY	MAPPING-TABLE
5	MAPPED-OP-CODE	MAPPING-TABLE-ENTRY
5	MAPPED-RELATIONS	MAPPING-TABLE-ENTRY
5	MAPPED-RECORDS	MAPPING-TABLE-ENTRY
5	MAPPED-SETS	MAPPING-TABLE-ENTRY
1	SETS-MAPPED	
3	SET-NAME	SETS-MAPPED
1	RECORD-MAPPINGS	
3	OP-CODE	RECORD-MAPPINGS
3	RECORD-NAMES	RECORD-MAPPINGS

PAGE NO. 00006 STOREMAP Data Dictionary

LEVEL	VARIABLE NAME	PART OF VARIABLE
3	SET-OR-AREA	RECORD-MAPPINGS
1	ATTRIBUTE-MAPPED	
3	ATTRIBUTE-NAME	ATTR TE-MAPPED
3	ATTRIBUTE-DISP	ATTR -MAPPED
3	FIELD-DISP	ATTRIBUTE-MAPPED
3	FIELD-LENGTH	ATTRIBUTE-MAPPED
3	FIELD-TYPE	ATTRIBUTE-MAPPED
1	RECORDS-MAPPED	
3	RECORD-NAMES	RECORDS-MAPPED
3	AREA-NAMES	RECORDS-MAPPED
3	CALC-FLAG	RECORDS-MAPPED
1	RELATIONS-MAPPED	
3	RELATION-NAME	RELATIONS-MAPPED

APPENDIX G

BATCH INPUT SYNTAX

STOREMAP

CARD TYPE	COLUMNS	DESCRIPTION
SUBSCHEMA	1-9	SUBSCHEMA
	11-26	(SUBSCHEMA NAME)
RECORD	1-9	RECORD
	11-26	(RECORD TYPE NAME)
	28-44	(AREA-NAME)
	45	(CALC FLAG 0 OR 1)
ATTRIBUTE	1-9	ATTRIBUTE
	11-27	(ATTRIBUTE NAME)
	29-31	(FIELD DISPLACMENT)
	33-35	(FIELD LENGTH)
	37	(SIGN FLAG S OR SPACE)
	38	(NUMBER OF WHOLE DECIMAL PLACES)
	39	(DECIMAL FLAG V OR SPACE)
	40	(NUMBER OF DECIMAL PLACES)
	41-43	(COMP TYPE C, C1, C2, OR C3)
SET	1-9	SET
	10-25	(SET NAME)

MAPPING	1-9	MAPPING
	11	(OP-CODE)
	13-28	(RECORD NAME)
	30-46	(SET NAME)

NETTOREL

APPENDIX H

TABLE LIMITATIONS

RELATIONS PER DATABASE	10
SETS PER RELATION	10
RECORD TYPES PER RELATION	10
ATTRIBUTES PER RECORD	10
MAPPINGS PER RECORD	10

APPENDIX I

TEST PLAN

STOREMAP

SYNTAX

VALID CARD TYPES

SUBSCHEMA

RELATION

SET

RECORD

ATTRIBUTE

MAPPING

END

INVALID CARD TYPE

ANY OTHER

MISSING CARD TYPE

SUBSCHEMA

RELATION

SET

RECORD

ATTRIBUTE

MAPPING

END

LOGIC

ADD CARD TYPE ALREADY PRESENT

SUBSCHEMA

RELATION

SET

RECORD

ATTRIBUTE

MAPPING

END

NETTOREL

SYNTAX

VALID CARD TYPE

SUBSCHEMA

QUERY

RESULT FILE

INVALID CARD TYPE

ANY OTHER

LOGIC

QUERIES

SELECT

PROJECT

JOIN

MULTIPLE SELECT

MULTIPLE PROJECT

MULTIPLE JOIN

SELECT AND PROJECT

SELECT AND JOIN

SELECT, PROJECT AND JOIN

SELECT, JOIN AND PROJECT

PROJECT AND SELECT

PROJECT AND JOIN

PROJECT, SELECT AND JOIN

PROJECT, JOIN AND SELECT

JOIN (ONE OWNER TWO MEMBERS) AND SELECT

JOIN (ONE OWNER TWO MEMBERS) AND PROJECT

JOIN (ONE OWNER TWO MEMBERS), SELECT AND PROJECT

JOIN (ONE OWNER TWO MEMBERS), PROJECT AND SELECT

JOIN (ONE MEMBER TWO OWNERS) AND SELECT

JOIN (ONE MEMBER TWO OWNERS) AND PROJECT

JOIN (ONE MEMBER TWO OWNERS), SELECT AND PROJECT

JOIN (ONE MEMBER TWO OWNERS), PROJECT AND SELECT

VITA

Captain Debra J. Nicely was born on 9 March 1954 in Albuquerque, New Mexico. She graduated from Washburn Rural High School in Topeka, Kansas in 1972. She graduated Cum Laude with a Bachelor of Science Degree in Mathematics from Troy State University, Troy, Alabama in 1977. She was a Distinguished Graduate of Officer Training School in September of 1977. She served as section chief of the application software section in the Directorate of Material Management at McClellan Air Force Base, California prior to her assignment to the School of Engineering at the Air Force Institute of Technology. She is married to Captain Robert E. Wilson.

Permanent Address: 1201 Money Street
Augusta,
Kansas 66701

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/GCS/EE/83D-15	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: School of Engineering; 6b. OFFICE SYMBOL (If applicable): AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code): Air Force Institute of Technology, Wright-Patterson AFB, Ohio 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS. — PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification): See Box 19	
12. PERSONAL AUTHOR(S): icely, Debra Jo B.S., Capt, USAF	
13a. TYPE OF REPORT: MS Thesis; 13b. TIME COVERED: FROM _____ TO _____; 14. DATE OF REPORT (Yr., Mo., Day): 1983 December; 15. PAGE COUNT: 117	
16. SUPPLEMENTARY NOTATION	Approved for public release: IAW AFR 190-17. Lynn Wolaver 7 Feb 84 LYNN E. WOLAVER Dean for Research and Professional Development Air Force Institute of Technology (AIC) Wright-Patterson AFB OH 45433
17. COSATI CODES: FIELD 09, GROUP 02, SUB. GR.	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Database, Mapping Between Databases, CODASYL, Network Database, Relational Database

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: THE OPERATOR MAPPING BETWEEN RELATIONAL ALGEBRA OPERATORS AND CODASYL BASED DATABASES MANAGED BY A CODASYL DBMS

Thesis Chairman: Charles Lillie, Major, USAF

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: Charles Lillie, Major, USAF	22b. TELEPHONE NUMBER (Include Area Code): 513-255-3576; 22c. OFFICE SYMBOL: AFIT/ENG

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED

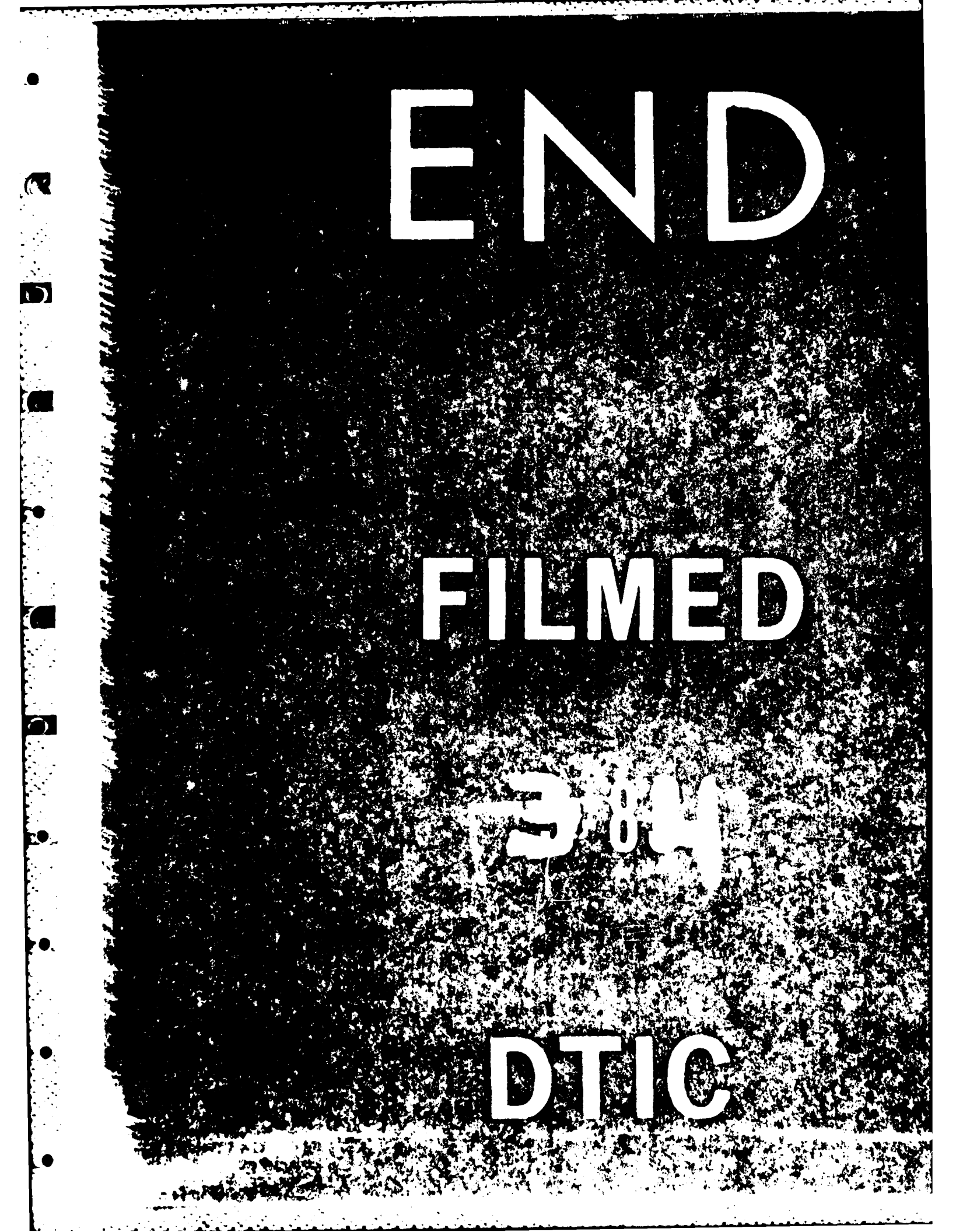
SECURITY CLASSIFICATION OF THIS PAGE

This thesis is an example of a mapping of a relational algebra query onto a network database. It consists of the requirements, definition, design and implementation of two generic COBOL programs for implementing such a mapping.

The first program STOREMAP uses a batch input file to build a data dictionary, on the original network database, which defines the relations on which relational algebra queries may be made. This input file is created by the Data Base Administrator who is the most knowledgeable of the structure of the network database and the relations which would be useful to the database's users.

The second program NETTOREL uses the defined relations in the data dictionary and relational algebra queries created by a user to generate a result relations. Data to be included in a result relation is determined by the data dictionary's definition of the relations contained in an associated query and the criteria set by that query.

This original effort shows that the theory for such an operator mapping is valid. Further efforts would be needed to make this implementation user friendly and therefore useful.


END
FILMED
DTIC